Security design and implementation

MORPHEMIC

Modelling and Orchestrating heterogeneous Resources and Polymorphic applications for Holistic Execution and adaptation of Models In the Cloud

H2020-ICT-2018-2020

Leadership in Enabling and Industrial Technologies: Information and Communication Technologies

Grant Agreement Number

871643

Duration

1 January 2020 – 31 December 2022

Deliverable reference

D4.2

Date

30 June 2021

Responsible partner

Engineering

Editor(s)

Ciro Formisano

Reviewers

Marta Różańska, Kyriakos Kritikos

Distribution

Public

Availability

www.morphemic.cloud

Executive summary

Security is a critical aspect in complex and heterogeneous platforms like MORPHEMIC: in such a context, the security by itself assumes complex roles definitions. The *Platform Security* includes all fundamental aspects, like Authentication, Authorization and Access Logging, aimed to preserve data and services in the *"internal"* operations. The authentication assures both that users are correctly identified and that the platform, acting on behalf of these users, is identified by the Deployment Environments. The authorization is designed according to the *Attribute Based Access Control* (ABAC) model. It is applied in two cases: *Role-Based User-Authorization*, for the basic user operations; *Pre-Authorization* to decide who, where and when is allowed to access a certain Deployment Environment. Finally, a mechanism of access logging keeps track of all the authentication and authorization attempts made in and by the platform.

In order to assure the highest security level to deployed applications, the selection of hosting nodes and environments must take into account their security features: this capability is defined as *Deployment Security*. The security features are formalized as security *parameters* and concur to the definition of the overall Utility.

MORPHEMIC includes components from previous European projects. The the document reports the results of an analysis on the reused components

The design of the MORPHEMIC's Security system is complete and consolidated: however, the implementation and the integration of all the related aspects will embrace the whole project period. Each component must be *secured* and integrated into the Platform Security system. The Deployment Security will proceed along with the other aspects related to the handling of a polymorphic application's Utility. The security mechanisms and guidelines provided should drive the whole development activity. However, it is possible that new functionalities will be needed, and new security requirements raised. This could happen during or even after the Project and could make it necessary to improve the security design and implementation.

Author(s)

Ciro Formisano (ENG), Daniele Pavia (ENG), Pawel Skrzypek (7BULLS), Ali Fahs (ACT), Ioannis Patiniotakis (ICCS)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871643

Revisions

Date	Version	Partner	Description
04/06/2021	0.1 (draft)	ENG	First draft
15/06/2021	0.2	UiO	First Review
16/06/2021	0.3	ENG	Revised version after the first review
03/08/2021	1.0	ENG	Final version after the review by FORTH and the members of the PMB

Table of Contents

Index of Figures

Index of Tables

Acronyms	
AAA	Authentication, Authorization and Accounting
ABAC	Attribute Based Access Control
AOP	Aspect Oriented Programming
BPM	Business Process Management
BYON	Bring Your Own Node
CAMEL	Cloud Application Modelling and Execution Language
CLI	Command Line Interface
CP	Constraint Programming
CRUD	Create Read Update Delete
DB	Database
DE	Deployment Environment
ESB	Enterprise Service Bus
GDPR	General Data Protection Rule
GUI	Graphical User Interface
IDS	Intrusion Detection System
IaaS	Infrastructure as a Service
IT	Information Technology
JSON	JavaScript Object Notation
JWT	JSon Web Token
LDAP	Lightweight Directory Access Protocol
MDS	Meta Data Schema
OASIS	Organization for the Advancement of Structured Information Standards
OS	Operating System

Acronyms	
PAMR	ProActive Message Routing
PDL	Policy Decision Logger
PNPS	ProActive Network Protocol over SSL
RBAC	Role Based Access Control
SAL	Executionware. Specifically, the abstraction layer (AL) that schedules (S) the application reconfiguration actions
SSH	Secure SHell
TOSCA	Topology and Orchestration Specification for Cloud Applications
UI	User Interface
URL	Uniform Resource Locator
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
VM	Virtual Machine

1 Introduction

1.1 Scope

This deliverable is focused on the design of the security system of the platform and its implementation. Security must be designed based on the provided platform functionalities and on the related risks, especially in terms of data. It is similar to a tailored suit, that must be adapted to the physical structure of who will wear it. The MORPHEMIC platform inherits some functionalities from MELODIC: for these functionalities data is used in similar ways. However, the security risks introduced by the new functionalities and the extensions of the legacy ones are prominent. The *polymorphism* extends the range of potential Deployment Environments, including also insecure nodes, while the heterogeneous supported application forms generate new risks due to the increased complexity of the deployment model. The *profiling functionality,* used to better define the deployment requirements of a certain application, needs an extended set of data, often coming from external sources. According to this, the basic security functionalities, authentication and authorization, inherited from MELODIC, must be reviewed and, if needed, extended. In addition, the introduction of an efficient mechanism of access logging is considered very useful to record the security events that may happen in this complex environment. This functionality becomes very important in case of attack: in fact a detailed analysis of the security-related events simplifies very much the work of who is in charge to find the cause and implement the countermeasures.

The heterogeneous Deployment Environments extend the *range of offers* available to the use cases. The *Utility* of the various offers is evaluated according to the parameters shown in the Deliverables *D2.3 ProActive Utility: Framework and Approach* and *D3.3 Optimized Planning and adaptation approach*. However, the analysis of the industrial requirements available in Deliverable *D6.1 Industrial requirements analysis* highlights the perceived importance of security also in terms of *Utility*. This suggested the introduction of the new concept of *Deployment Security*, which represents the capability to select the most suitable Deployment Environment considering its security level along with the other parameters used to evaluate the Utility. The Deployment Security introduces several challenging aspects on all the functionalities of MORPHEMIC: for instance, the evaluation of the *infrastructure* or *node security features* or the *selection* of the most valid one for a certain component by matching the specific security features against the security requirements.

This Deliverable provides detailed description of all the aforementioned security-related functionalities and aspects.

1.2 Target audience

According to the DoW, this Deliverable is a public document. All the sections are aimed at a technical audience. Specifically, Chapters 2, 3 and 0 include architectural descriptions appropriate for technical architects specialized in security and infrastructures, but even developers can benefit from them. Chapter 5 is focused on the components inherited from other projects, specifically MELODIC and MARKOS. Their security features are analysed and some suggestions to adapt them to MORPHEMIC are provided: developers and component owners could benefit from this chapter. The document, especially its first part, contains a comparison between MELODIC and MORPHEMIC, aimed to specify what has been fully or partially inherited from MELODIC and what has been newly introduced into MORPHEMIC: these aspects are very useful for the members of the consortium, especially developers and component owners, that were not part of the MELODIC project.

All the aforementioned aspects are useful for an *internal* audience, i.e., members of the project consortium. The deliverable aims to be a guideline to develop secure components to be integrated in a secure platform. However, some technical and architectural concepts, such as the architecture and technology of the authorization system (Section 3.2.1), can be interesting also for external architects and security experts. Finally, the concept of Deployment Security (Chapter 4) can be a research topic that researchers can extend.

1.3 Related Deliverables

The work documented in this Deliverable is related to other deliverables of MORPHEMIC. The deliverables that will be mentioned are:

- Deliverable *D4.1 Architecture of pre-processor and proactive reconfiguration*: concerning the architecture, processing flow and description of the mentioned components

- Deliverable *D5.3 Deployment artefact manager*: concerning the details on the Scheduling Abstraction Layer and ProActive Scheduler and Resource Manager, specifically on the association of a new Deployment Environment (*addCloud* method) and the related profile to be extended in terms of security
- Deliverable *D6.1 Industrial requirements analysis*: concerning the requirements to which the work on security is referred.

1.4 Structure of the document

The Deliverable presents the following structure:

- Chapter 2, Security concepts, describes the theoretical aspects that are the basis of the security architecture of MORPHEMIC. It also includes the security-related requirements of Deliverable *D6.1 Industrial requirements analysis* as well as defines the concepts of Platform Security and Deployment Security
- Chapter 3, Platform security's functionalities and architecture, describes the functionalities related to the Platform Security, namely Authentication, Authorization and Access Logging
- Chapter 4, Deployment Security, is focused on the concept of Deployment Security, how it is applied in MORPHEMIC and which architectural components implement it
- Chapter 5, MORPHEMIC's reused components: security analysis, provides detailed information on the security characteristics of the components of the current release of MORPHEMIC and the next steps planned to improve it.
- Chapter 6 summarises the document and supplies a set of final considerations.

2 Security concepts

There is not any universally accepted definition of security. There is also a debate around the difference between *information security* and *cybersecurity*, which are often considered synonymous.

According to the definition of the SANS institute[1], "*Information Security refers to the processes and methodologies which are designed and implemented to protect print, electronic, or any other form of confidential, private and sensitive information or data from unauthorized access, use, misuse, disclosure, destruction, modification, or disruption*"[2]. Cybersecurity is a broader practice, focused on *defending IT assets from attacks*: it is a sort of *parent* of information security, while *network security* and *application security* are *brother disciplines*[3].

Cybersecurity is very wide and extends over different levels much beyond secure code and information security: it includes also the physical accesses to server rooms and training of the operating personnel. Application security is focused on the code of the application and on the measures to make it *secure*.

Topics such as *server room security* or *personnel training* are out of scope of this document: in the MORPHEMIC project the priority is to secure the information/data stored in the platform and to assure that the selected Nodes and Deployment Environments provide security features adequate to the security requirements of the associated applications. For this reason, according to the aforementioned definitions, the most appropriate one in the context of MORPHEMIC is *information security*, but some aspects of *application security* will be discussed, especially concerning the inherited components.

The starting points are the MELODIC's Security system, described in the Deliverable 5.3 of MELODIC [1], and the industrial requirements, listed in Deliverable *D6.1 Industrial requirements analysis*.

MELODIC is a multi-cloud deployment platform, which supports the optimized deployment of applications on nodes selected from different cloud providers. The platform supports two information flows: from the user to the platform and from the platform to the clouds. For both flows, an authentication and an authorization mechanism are provided.

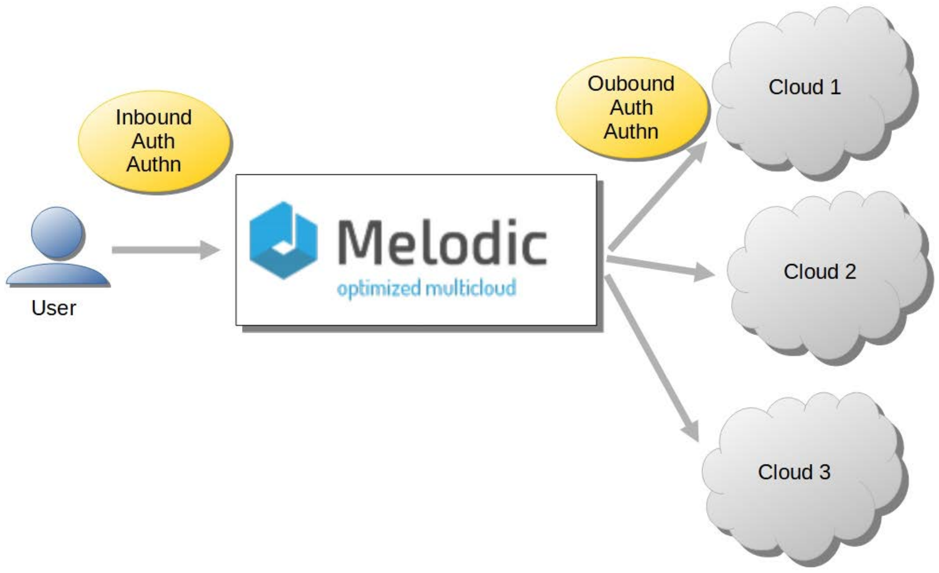

Figure 1 MELODIC's inbound and outbound security

[1] https://www.sans.org/
[2] https://www.sans.org/information-security/
[3] https://www.csoonline.com/article/3513899/what-is-information-security-definition-principles-and-jobs.html

The user produces data (e.g., registration data, infrastructure ownership, etc.) and requests for services (e.g., application deployment): this requires that his/her identity is assured (authentication) along with the privileges to do what he/she requests (authorization). These processes are identified in Figure 1 as "*inbound authentication and authorization* and do pursue this aim. Most of the cloud infrastructures need authentication and authorization: in this case the MELODIC platform acts as a client using appropriate credentials (available in an internal secure storage) to access them. Furthermore, some extra authorization policies, to be enforced by the MELODIC platform, guarantee a finer grained control on infrastructures access. The latter mechanisms are identified as *outbound authentication and authorization* in Figure 1. MORPHEMIC interacts with users and deployment environments in similar ways, so the rationale is preserved. However, the polymorphic adaptation makes it necessary to extend the concepts and to update the supporting technology (Chapter 3).

The industrial requirements outline the need of two levels of security: the first one related to the platform and the access to all the provided services, including Deployment Environment management; the second one guarantees for the deployed application that the security level provided by the Deployment Environment is adequate with respect to the security requirements of the application.

The industrial requirements related to security are listed in Section 2.1, while Sections 2.2 and 2.3 describe the aforementioned kinds of security. Specifically:

- the *Platform Security* includes access control mechanisms, privacy, and confidentiality for user data and the capability to prevent unauthorized access to the services offered to a specific user, along with the capability to log all the security related operations
- the *Deployment Security* is the capability to select the most appropriate Deployment Environment and application form according to the security requirements of the application: it has strong dependency on the associated infrastructures representing the offer that MORPHEMIC will take into account for the application (deployment & adaptive provisioning).

2.1 Security related requirements

The industrial requirements of MORPHEMIC, listed and described in Deliverable *D6.1 Industrial requirements analysis*, make explicit the needs of the users in terms of security. Table 1 summarizes them.

Table 1 Security related requirements in Deliverable 6.1

ID	Description	Category
MEL-8	Secure and context-aware data access control mechanism	Platform Security
UC-C-SEC.1	Support for traffic isolation	Infrastructure
UC-C-SEC.2	Support for secure communications	Application-related
UC-C-SEC.3	Support for security-related applications	Security-management-related services
UC-1.SE.8	Support for Security Management	Security-management-related services
UC-2.SE.2	Support for private execution environment	Infrastructure

The requirement MEL-8 is derived from MELODIC and confirmed in MORPHEMIC. Specifically, it assures that MORPHEMIC offers "*the appropriate tools that guarantee the secure access and processing of all the involved data, handled by big-data intensive applications deployed and maintained in multi-clouds: any user should be properly* *authenticated before accessing data that reside in Cloud infrastructures*". The description embraces all the aspects of the Authentication and Authorization for a platform managing a multi-cloud environment. The requirement is very complex: the related aspects are made more explicit in Deliverable 5.3 of MELODIC [1]. Since MORPHEMIC extends the multi-clouds support to Polymorphic Adaptation, also the enhancement of the related security functionalities should be extended to cover the novel aspects.

The Platform Security is mainly finalized to meet this requirement and all its implications on MORPHEMIC features. Concerning the design, the Authentication and Authorization are derived from and extend the corresponding functionalities of MELODIC. Concerning the technology, some improvements have been introduced to increase the effectiveness and the consistency of the whole process (Section 2.2).

The remaining requirements of Table 1 are focused on the security characteristics that the associated Deployment Environments must provide. The three use-cases of MORPHEMIC are heterogeneous and embrace very different contexts: the security requirements of their applications are different but, in general, highlight the perceived importance of the Deployment Environment in this sense. Part of the listed requirements concern the supported environments, for example, private clouds or on-premises. The motivation of these requests is related to security, since some legal constraints (e.g., GDPR) or simple perception, suggest that some infrastructures or geographical locations are more secure than others. Other requirements are focused on the security related services that some Deployment Environments natively provide, such as Intrusion Detection systems. It is remarkable that this requirements group does not represent a potential challenge for MORPHEMIC for two main reasons:

1. in most cases these services are supplied by the infrastructure provider
2. in case they are not supplied or if it is necessary to deploy appropriate clients, they should be considered by MORPHEMIC as deployable applications and, according to the requirements, every potential application or service must be *deployable* by MORPHEMIC.

Concerning the *intrinsic* features of the Deployment Environments, such as the type of environment or geographical location, some considerations are very interesting:

- each associated Deployment Environment or node candidate will present part of the required features, but not all: for example, it can be *located* in Europe but not in USA, can be a BYON (Bring Your Own Node), belonging to a cloud or physical, public or private infrastructure. This means that the specific security requirements of a certain application or service, must be matched with the specific features of the proposed infrastructure or node candidate
- as a consequence of the previous point, the security characteristics of a certain Deployment Environment or node will impact the selection of the offered nodes candidates as well as the overall *Utility* that can be achieved by the application.

The Deployment Security (Section 2.3) is the answer to these considerations. MORPHEMIC will take into account also the security requirements of the components of the deployed application to define a deployment configuration. Therefore, the Utility will also include security related parameters.

2.2 Platform Security

The core functionalities of the MORPHEMIC platform are founded on MELODIC. They are integrated with the new ones concerning Polymorphic Adaptation, Self-Healing and Hardware Acceleration, which require a revision of MELODIC Security system to define where and how it has to be modified or extended.

At a glance, both the platforms enable users to deploy applications on certain environments, so the core functionalities concerning security are the same. Specifically, users must be *authenticated* to access the platform and *authorized* to use specific resources. As *resource* is intended both for the provided functionalities (e.g., *user management*, *infrastructure association,* etc.) and the Deployment Environment (e.g., the instance of AWS belonging to a certain user or a specific Edge Node).

The new functionalities of MORPHEMIC introduce the need of extra care at the technological level and in terms of *infrastructure hardening* (Chapter 5). The technological details of the Platform Security will be described in Chapter 3, for the Authentication and Authorization systems and for the new access logging functionality. Moreover, Chapter 5 will provide a more detailed technological analysis of the components that MORPHEMIC inherited from other projects. These components must be adapted to support the new functionalities and the overall objectives of the new platform: this includes also an analysis in terms of security to drive the subsequent development activities. Specifically, the components inherited by MELODIC represent the core of the basic functionalities of MORPHEMIC and should be improved in terms of generic security (i.e., open ports, proxying etc.). Furthermore, the Web Crawler, inherited from the MARKOS[4] project, must be analysed in order to understand if its security features are up-to-date and suitable for the required interactions with heterogeneous external data sources.

The capability to log *access related information*, including access attempts, the involved resources and the results (Section 3.3) represents the third functionality of the Platform Security. The Access Logging system provides useful information to have a consistent control of the security status of the MORPHEMIC platform and to get data to identify the causes of possible attacks and prepare the countermeasures.

2.3 Deployment Security

The MORPHEMIC platform is conceived to be associated with different Deployment Environments and deploy there every application. In general, applications have specific security requirements that restrain the potential Deployment Environments. The application requirements represent the sum of the requirements of the services and/or components that constitute it, and could be related to data confidentiality, access control, intrusion detection, etc. In general, it is possible to map the security requirements of a certain component to the node that hosts it through the CAMEL Model. The mapping can be implicit or explicit: in the first case ,the Application Manager should include somehow in the Camel Model the security requirements for each component. In other terms, for each node it should be explicitly indicated, for example, the needed firewall settings or the required access control mechanisms. It is easy to understand that this would significantly increase the complexity in the CAMEL Model and make more difficult the task of the Application Manager.

The alternative solution consists in implicitly associating the security requirements to the component/service to be deployed. For instance, let us consider an application which includes a web-GUI. It is possible to derive the following security-related requirements for such an application component:

- the user interface should be exposed on port 80, which should be open on a public IP address
- if possible, all the other needed services (e.g., SSH) should be available on a private network
- the minimum possible services should be enabled
- SELinux, along with a specific configuration of the firewall, should be available and configured according.

These security requirements are valid for most of the web-GUIs: this example is valid in general, i.e., in most of the cases, the security requirements are strictly related to the provided service. The implicit association between security requirements and component/service, consists in the transparent application of the *default* security requirements for the deployment of each component specified in the CAMEL Model. This simplifies the configuration steps in charge of the Application Manager, providing, in most of the cases, the same result in term of security. This solution appears to provide the best trade-off between configurability and security and has been chosen for the MORPHEMIC Deployment Security.

In particular, the Deployment Security is founded on two complementary aspects:

- the security level provided by the Deployment Environment, defined by its *security profile* (described in Section 4.2), expressed as *security parameters*
- the security level requested by the application, defined by the *security requirements* of the application components, specified in the CAMEL model, which are implicitly expressed in form of *security parameters* as well: in other terms, a Database requires that its hosting node provides certain parameters, a GUI others etc.

These parameters are used in the Utility Function in order to compare what is required by the application and what is provided by the Node Candidates and, finally, try to maximize the overall Utility considering also security.

As conceived, the detail of the Deployment Security will be transparent to the Application Manager, since he/she will have to define only the required nodes and functionalities of the application: the related security requirements are implicitly associated to the nodes and will be matched with the security profile of the hosting node.

The associated Deployment Environments and their nodes should present specific security profiles mapping their security features. Even if some research topics concerning the classification of Deployment Environments (especially

[4] https://cordis.europa.eu/article/id/117369-markos-global-integrated-and-searchable-opensource-software

cloud infrastructures) in terms of security exist (Section 4.1), an automatic classification has not been conceived so far, and its design is out of the scope of MORPHEMIC. Therefore, security profiles will be manually associated to each Deployment Environment or Node by the *Infrastructure Manager* (Section 3.2.1). For some well-known infrastructures (e.g. AWS) the security profiles could be at least partially pre-determined along with the specific features of the infrastructure: for other cases, including Edge or on-premises nodes, they will be manually defined through the process described in Section 4.3. The security profile of a certain Deployment Environment is automatically inherited by all of its nodes; however, the Scheduling Abstraction Layer and the ProActive Scheduler will enable to define multiple security profiles on a single Deployment Environment according to the different available security features. In practical terms, this means to (manually) define a sort of *default security profile* for a certain Deployment Environment, containing the minimal provided security features, and a set of *extended* security profiles to override the default one and include the more *extended* capabilities (Section 4.3).

3 Platform security's functionalities and architecture

The Platform Security assures that the MORPHEMIC platform provides services with an adequate security level and guarantees that the data are securely stored. The core functionalities are *Authentication* (Section 3.1), *Authorization* (Section 3.2) and *Access Logging* (Section 3.3). Figure 2 reports the MORPHEMIC's architecture, defined in the Figure 11 of Deliverable 4.1 (*Architecture of pre-processor and proactive reconfiguration*): the module in blue (called Security Module, in that Deliverable), includes all the functionalities described in this section.

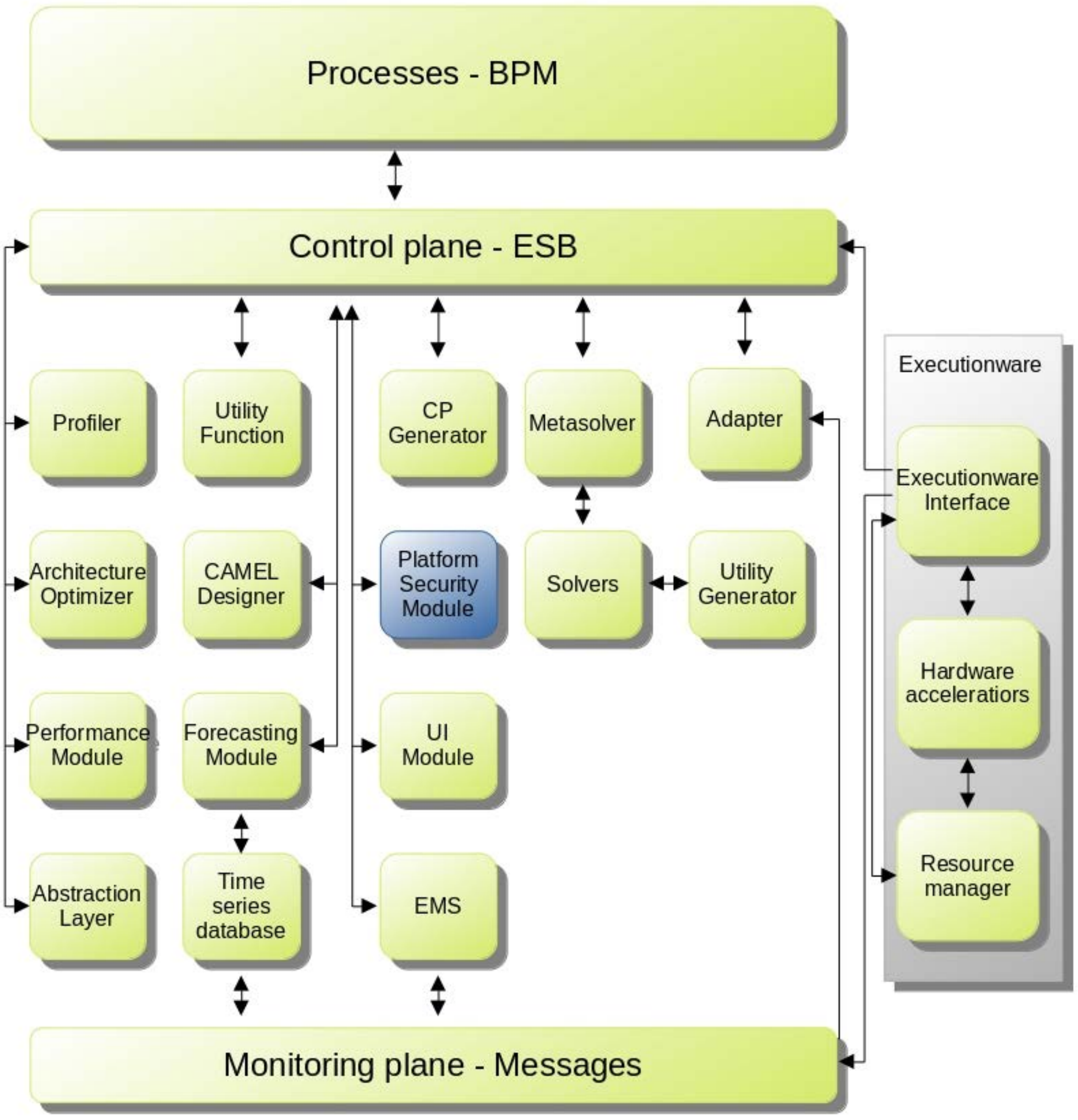

Figure 2 Architecture of the MORPHEMIC Platform

The Platform Security Module represents a *cross component* functionality: the actual implementation of the Authentication, the Authorization and the Access Logging is distributed to different components. In particular the MORPHEMIC platform provides services implemented by some components, such as the Utility Function Creator, the Adapter, and the Solvers. The basic architecture of the Authentication mechanism is derived from MELODIC and is based on a JWT Token: specifically, each *request for a service* must be associated to a *token* encoding the identity of the requester (Section 3.1). All the components perform at least the token validation as a minimal authentication check, while a subset of them also applies Authorization enforcement.

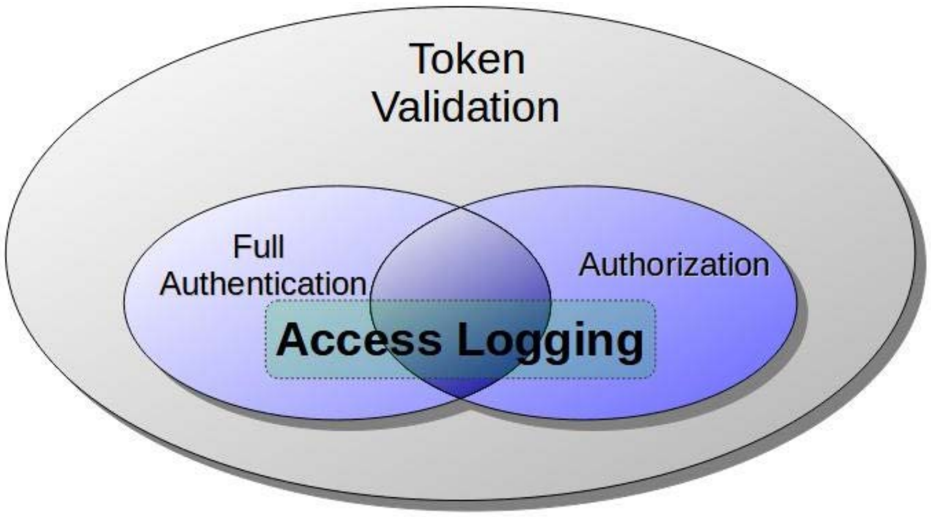

Figure 3 Platform security functionalities

More in detail, the relationship between the operations of the Platform Security is depicted in Figure 3:

- the *Token Validation* is the minimal task related to Authentication: a valid token has not expired and refers to a valid identity
- the *Full Authentication* includes credentials verification, based on the token's content (if existing), or the generation of the token starting from a username/password pair: this task is performed once per-session by the *entry points* of the platform
- the *Authorization*, at component level, includes the enforcement of a *policy* associated to the attributes related to the *identity* obtained from the Token and possibly integrated with attributes related to the *environment* (Section 3.2).

The simple token validation is preliminary to all the security tasks: each request forwarded to each internal service of the MORPHEMIC architecture must be valid in term of issuer and expiration; otherwise, it must be discarded. Each service or component of MORPHEMIC must validate the token before executing the respective (service) operation. What is defined as "Full Authentication" is the verification of the user credential and the generation of the token: the credentials are set in the User Interface and the LDAP based Authentication system performs the validation. Part of the internal components of MORPHEMIC, identified through criteria exposed in Section 3.2, asks the Authorization system to *decide* if a certain operation can be executed according to the requester's privileges and the environment (date/time/location etc.). The components that perform Full Authentication or Authorization are integrated to the Logging system to keep track of the security related operations.

As mentioned before, Authorization decisions are requested only by some specific components and only Full Authentication and Authorization operations are logged (not the simple Token Validation). This should guarantee an acceptable trade-off between complexity (and performance) and security. In fact, the request for Authorization decision involves several components and is per-service. In other terms, a certain user that is *authorized* to access a certain infrastructure, is indirectly authorized to use all the related components (e.g., Scheduling Abstraction Layer, ProActive Scheduler, Adapter etc.): if an authorization decision is requested for each component, it will provide the same result and will introduce more latency. Furthermore, the services that require authorization are classified as *critical*, i.e., the risk generated by unauthorized operations is high. Concerning the Logging System, the possibility to log all the Token Validation operations would force the components to send logging messages to record if a certain token is still valid. In order to analyse potential attack patterns, the access attempts through authentication or the events concerning authorization privileges are much more valuable. For this reason, the increase in terms of complexity and latency to log the Token Validation operations is not justified.

The Platform Security is also guaranteed by the so-called *hardening*, which includes all the tools and measures to reduce the risks of external attacks. This aspect should be considered part of the Platform Security; however, since it has much more to do with the technology than the design, it will be discussed in Chapter 5 where the status of the components will be examined.

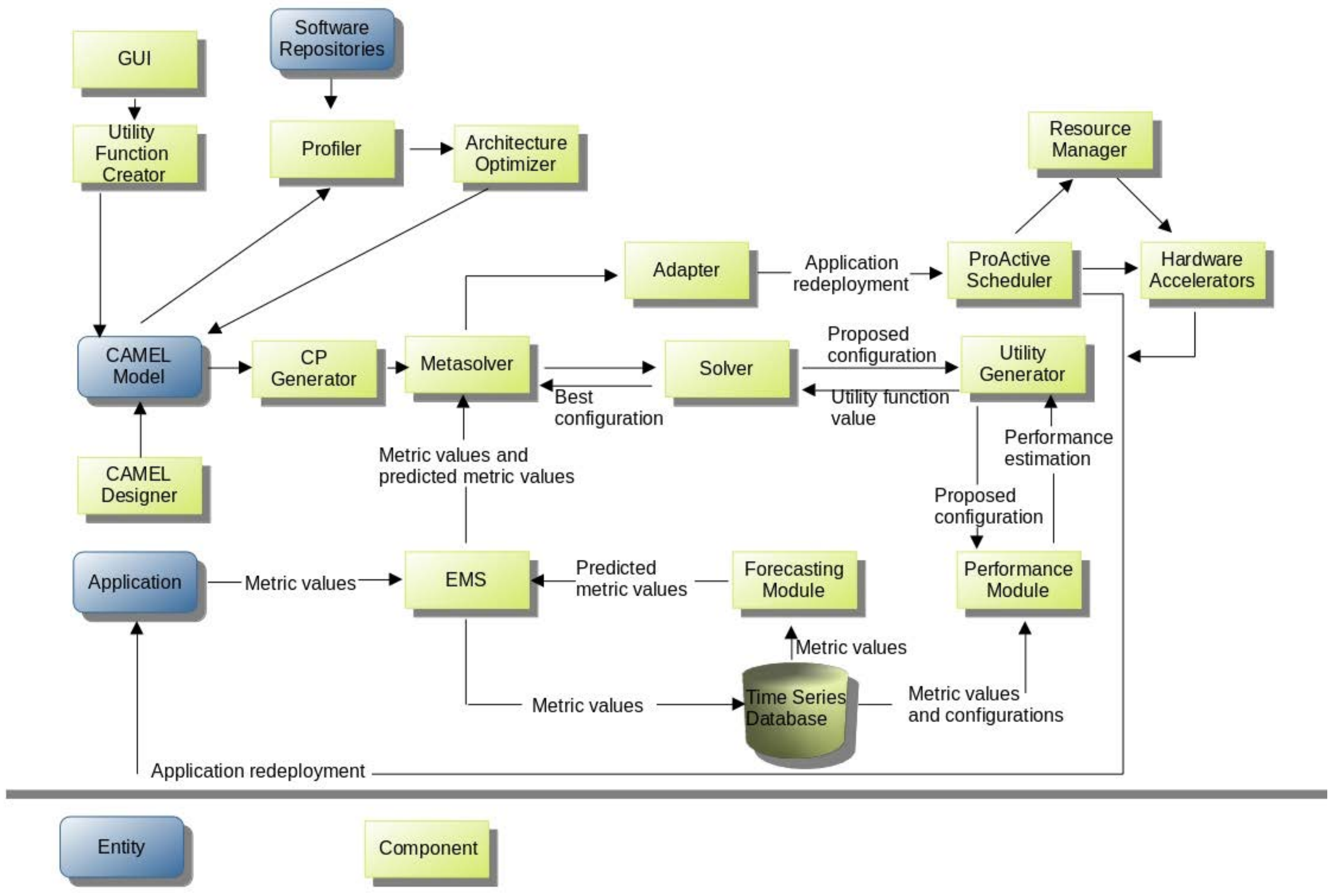

Figure 4 Key components of MORPHEMIC

Figure 4 shows the components and entities of MORPHEMIC: the components are part of the MORPHEMIC's architecture, and the entities are the resources which interact with it. The direction of the arrows represents the data flow while the paradigm used is request/reply. The minimal Token Validation includes checksum and lifetime verification, in order to prevent malicious external access and expired requests. The GUI provides controls suitable for human users, however, MORPHEMIC will provide also direct Web Services access. In both the cases, the authentication is requested to guarantee secure access: in order to easily express the login functionality provided by the access points, we will use the term *Login Client*. The Login Client will indicate in this document the functionality to get the user credentials (in different format, depending on the access point) and obtain the token (Section 3.1.1). This operation includes the Full Authentication: the requests for services coming from the external must always pass through the Login Client to be authenticated. This means that requests without any token or containing an invalid or expired one, discarded or, if possible, redirected to the Login Client.

The following sections will describe in detail the Authentication, Authorization and Access Logging mechanism of MORPHEMIC comparing them with the ones provided by MELODIC. Figure 5 shows all these functionalities: the blocks in green have been introduced in MORPHEMIC. The new blocks concerns mainly the Access Logging, which is the *new* functionality: however, the ABAC Authorization mechanism has modified its scope with respect MELODIC and, on this sense, can be considered *new* as well.

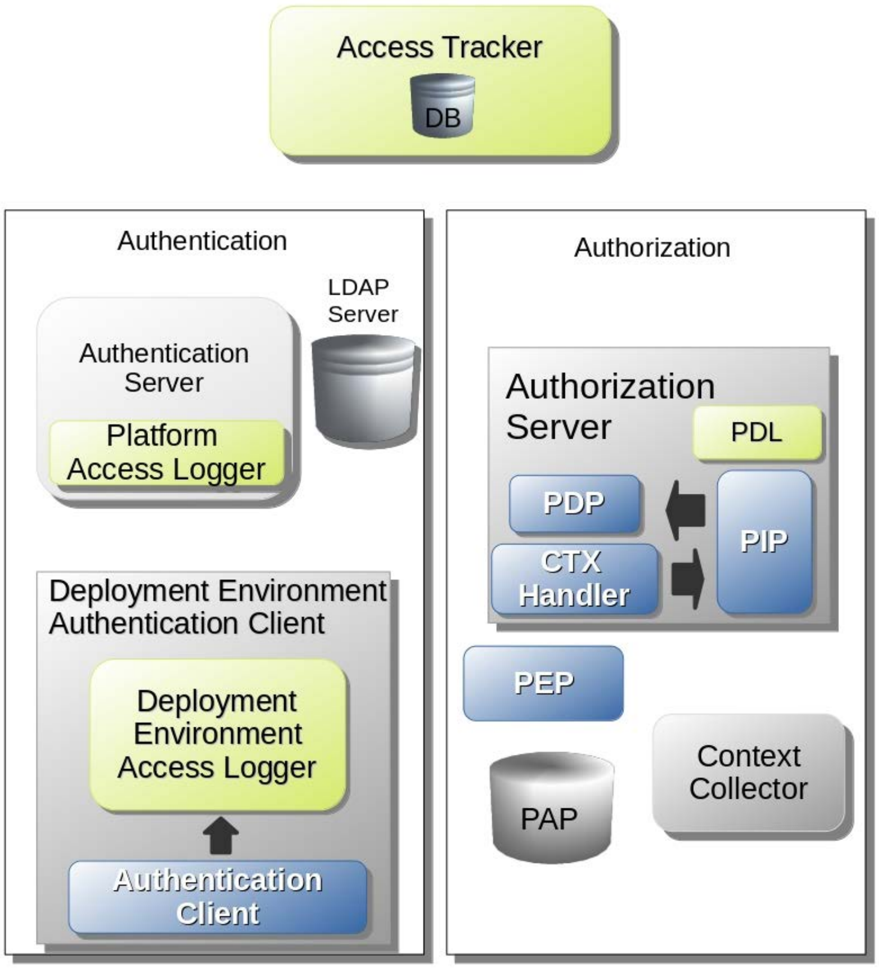

Figure 5 Complete Platform Security System in MORPHEMIC

3.1 Authentication system

Concerning the Authentication, the MORPHEMIC platform plays the role of both a client and a server according to the specific functionality requested:

- *Server* for the *User-Authentication* (Section 3.1.1), regarding *inbound* requests, to verify the identity of the users accessing the platform
- *Client* for the *Deployment Environment Authentication* (Section 3.1.2), regarding *outbound* requests, when the platform interacts with Deployment Environments and their authentication mechanisms: in these cases, MORPHEMIC must log into the external environments by using specific credentials, provided by the infrastructure owner during the configuration.

Both processes are supported by MELODIC as well: for this reason, some architectural aspects have been preserved. In particular, MORPHEMIC supports username and password authentication against accounts stored on an LDAP directory and the capability to internally store the credentials needed to access the Deployment Environment [1]. Concerning the technological aspects, the User-Authentication does not experience any major modification, while the Deployment Environment Authentication in MORPHEMIC is performed by the new *Executionware*, based on the *ProActive Scheduler*[5] that extends the range of supported Deployment Environments including, at the time of writing this Deliverable, Edge Nodes.

3.1.1 User-Authentication

The User-Authentication system of MORPHEMIC is derived from MELODIC: Figure 6 shows the generic flow.

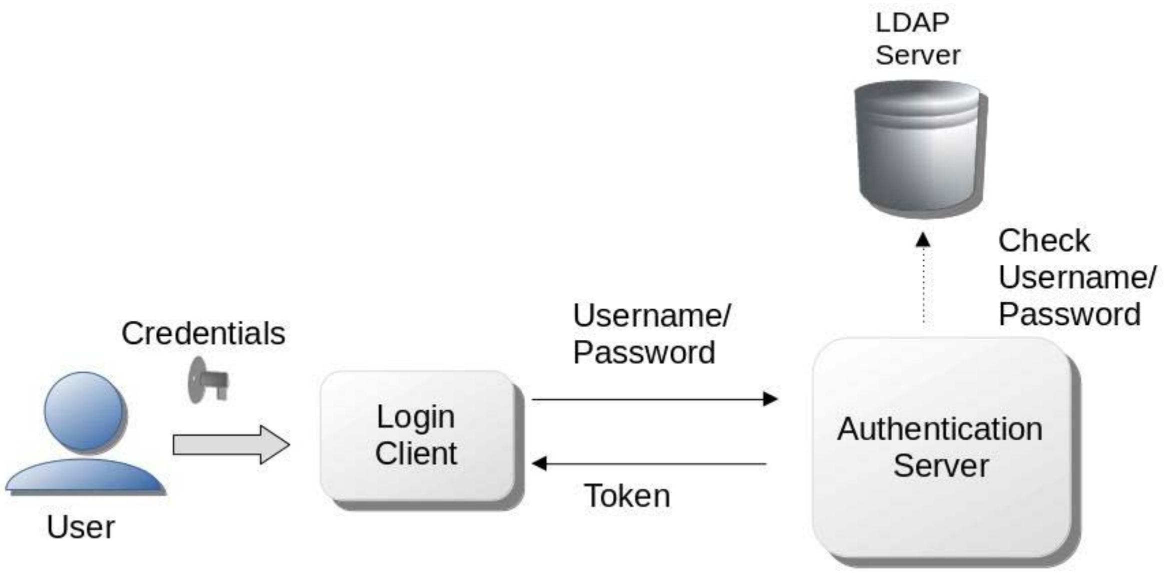

Figure 6 Authentication system

As mentioned before, the *Login Client* is the generic component where the user supplies his or her credentials: all the access points of MORPHEMIC, GUI and Web Services, exhibit this functionality. The credentials are then forwarded to the *Authentication Server*, checked against an LDAP, and the authentication is returned in form of a *Token*.

As the details of this process are described in [1], this section focuses on how this process is adopted by MORPHEMIC and which components exploit it.

The Authentication process produces an JWT Token ([2] and [3]) which is used to identify the request and is checked by the components that need authentication and authorization. The association is possibly automatic, performed transparently at GUI level by using the *session*. For stateless Web Services the token should be manually included in the *Authorization* header of the HTTP message.

[5] https://proactive.activeeon.com

Specifically, the payload of the token contains two mandatory pieces of information:

- the username of the requester (*subject*)
- the *expiration time*.

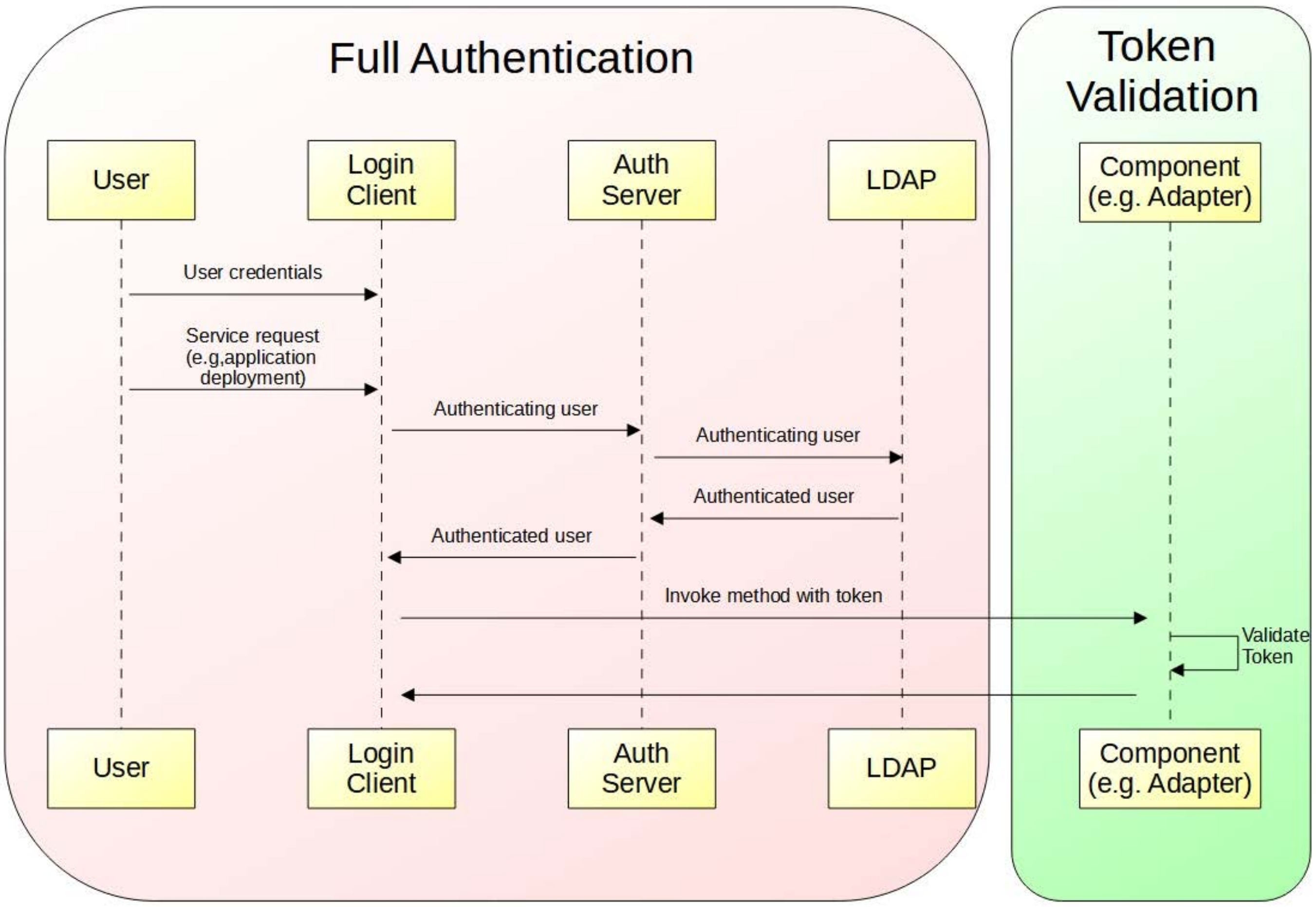

Figure 7 Authentication operations flow

Figure 7 shows the operations flow of the Authentication. The example contains the Full Authentication and Token Validation functionalities. The user sends a service request (e.g. application deployment); this can be done in two different ways:

1. through GUI, so after provided the credential and obtained a Token in the session
2. through Web Services, so by manually including the Token obtained after an independent authentication process.

In both the cases credentials and service request pass through the Login Client functionality which manages the Full Authentication Process by cooperating with the Authentication Server and the LDAP. The final result (of the successful authentication) is the Token which is included in the request that, in turn, is forwarded to the Component executing the operation. This component (e.g., the Executionware) validates the Token and executes the action.

The local Token Validation consists in the verification of the token *signature* and *expiration*: this reduces the risk to accept malicious or expired requests. It is important to remark that in this example the Authorization process is not considered: more in general, some components also verify the respective *Privileges* (Section 3.2).

3.1.2 Deployment Environment Authentication

In order to interact with the Deployment Environments, MORPHEMIC must comply with their security mechanisms. For instance, cloud providers require authentication to access their services and manage their virtual environments. In

MELODIC this was the only use case, explained in Chapter 7 of [1], where the credential storage module is described as well. In MORPHEMIC the context is much more generic: Deployment Environments are not only cloud providers, but also other sets of nodes, including Edge and isolated nodes provided according to the BYON paradigm.

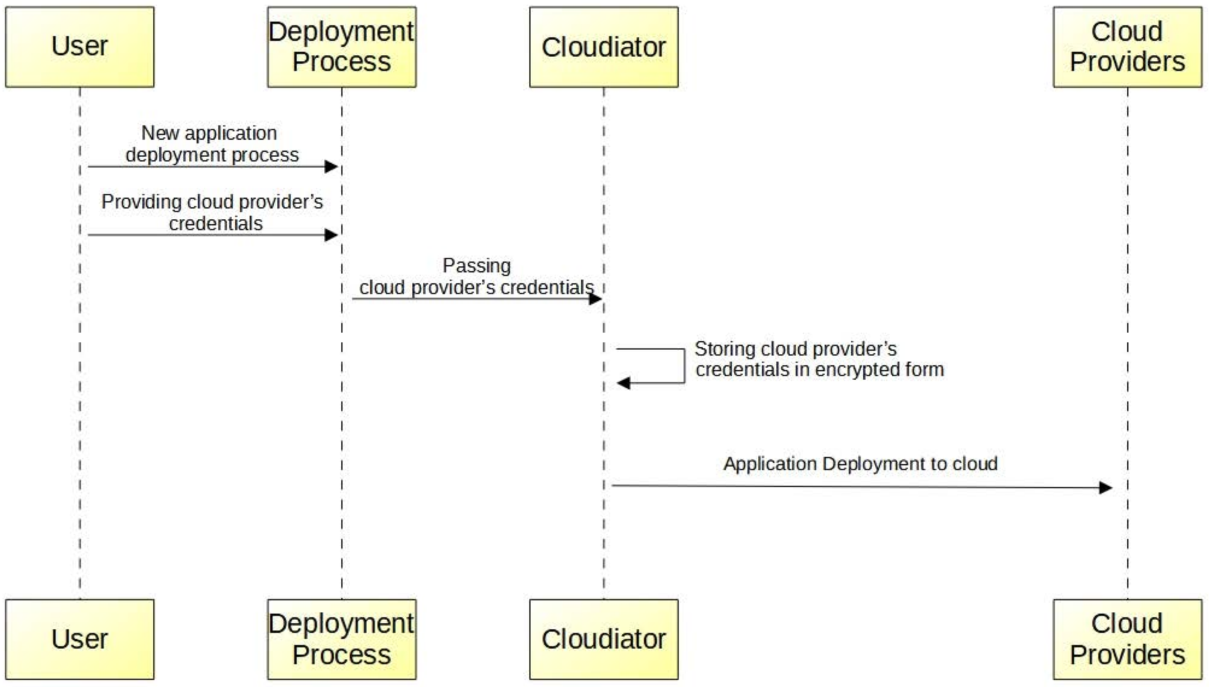
User
Deployment Process
Cloudiator
Cloud Providers
New application deployment process
Providing cloud provider's credentials
Passing cloud provider's credentials
Storing cloud provider's credentials in encrypted form
Application Deployment to cloud
User
Deployment Process
Cloudiator
Cloud Providers

Figure 8 MELODIC's Cloud Providers' credential flow

Figure 8 is identical to Figure 10 of [1] and shows the MELODIC's Cloud Providers' credential flow, which is generalized in MORPHEMIC by replacing Cloudiator with ProActive and the Cloud Providers with generic Deployment Environments (Figure 9).

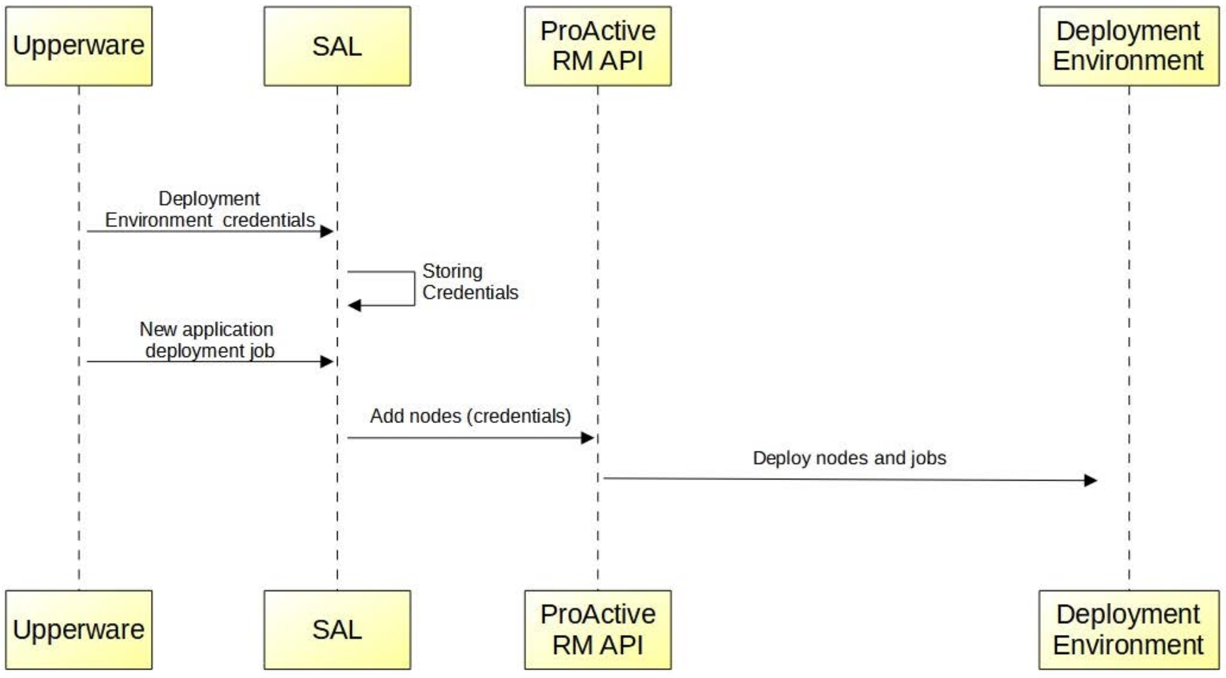

Figure 9 Deployment Environment credentials flow

The picture is detailed in Deliverable 5.3 (*Deployment artefact manager*), specifically, a new Deployment Environment is registered by the Executionware Access Layer (SAL) which stores its access credentials and include its nodes among the potential nodes candidates. The deployment process uses the stored credentials and leverages the ProActive Resource Manager (RM) to manage the Deployment Environment.

Cloud Providers support a wide set of authentication factors, such as username/password, certificates, source address, and multi-factor. The introduction of non-cloud environments extends this set, introducing also the use case of *open-access* environments, i.e., environments or nodes that do not support any kind of authentication. For instance, in some cases authentication does not make sense, such as when the user associates a proprietary hardware node in a protected environment. As defined by the requirements of Deliverable 6.1 (*Industrial requirements analysis*), MORPHEMIC will have to consider also non-Cloud Deployment Environments: as a consequence, the aforementioned security related configuration should be supported. The Deliverable 4.1 (*Architecture of pre-processor and proactive reconfiguration*) describes how the new *Executionware*, based on the ProActive Scheduler, will support them in general, while Section 3.1.2 of this Deliverable will give more details in terms of security.

In MORPHEMIC, the component that manages the application deployment is MELODIC: among the MELODIC's components, the *Executionware* enacts all the deployment decisions. The version of MELODIC used in MORPHEMIC replaces the original implementation of Executionware, Cloudiator[6], with the ProActive Scheduler. The ProActive Scheduler manages the Deployment Environment credentials for all the supported Deployment Environments, including Edge Nodes. It guarantees flexibility, efficiency and high-quality support, provided by a member of the project consortium, ActiveeOn. The ProActive Scheduler supports all the aforementioned use cases, included *open-access* environments and edge nodes: it provides a credential storage module as well. In terms of security, the replacement of Cloudiator with the new component did not impact the operation flow, which will be preserved, as shown in Figure 7: more generic information on the ProActive Scheduler is available in Deliverable 4.1 (*Architecture of pre-processor and proactive reconfiguration*).

3.1.3 Implementation details: Executionware – ProActive Scheduler

The ProActive Scheduler acts as a central entity that controls resources aggregated from heterogeneous environments like cloud providers, legacy servers, and edge devices. The ProActive Resource Manager is responsible for orchestrating and monitoring the connected devices.

The Resource Manager assures that the resources decided by the Upperware are available to the ProActive Scheduler. Here we have to draw a distinction between connecting resources coming from cloud providers that requires access authentication, and BYON resources that are connected directly from the nodes.

Once the MELODIC Upperware decides on using a certain Cloud provider, the Executionware (SAL) will be contacted to execute the decision. Consequently, the Executionware communicates with a component called Connector-IaaS7 that handles the communication with several cloud providers.

In definition, Connector-IaaS creates a single endpoint where all the communication with the infrastructure is unified. This improves flexibility and reduces the provider dependencies by allowing easy migration from one provider to another. The multi-IaaS connector enables to do CRUD operations on different infrastructures on public or private Clouds (AWS EC2, Openstack, VMWare, Docker, etc). These operations are conducted using Apache jclouds8 that provides a unified JSON format to connect to several cloud providers.

On the other hand, to connect BYON nodes to the Executionware, the owner of the nodes will provide the SSH credentials that will allow the Executionware to automatically connect the nodes to the Resource manager using ProActive SSH infrastructure. In this step, SAL creates a new user account on the BYON nodes and adds the server's public SSH key to the set of authorized keys for the newly created user. As a result, ProActive is granted full control of the nodes through SSH.

[6] https://github.com/cloudiator
[7] https://github.com/ow2-proactive/connector-iaas
[8] https://jclouds.apache.org/

3.2 Attribute based authorization

An authenticated user can access a subset of the functionalities of the MORPHEMIC platform: the functionalities for which he or she is *authorized*.

Each functionality is implemented by one or more *components* (Figure 4), whose access is limited to the users associated to certain *attributes* and in certain *environment* (context) conditions. The environment is by itself defined by some attributes; for this reason, the best authorization model appears to be the *Attribute Based Access Control* (*ABAC*) [4].

The ABAC model is very flexible since the attributes used to define policies can refer to different elements. In particular, the model supports per-application policies, aimed, for example, to permit the deployment of a certain application only on a specific Deployment Environment. This possibility can be useful for possible future interesting extensions of MORPHEMIC, which, at the time of writing this deliverable, is still *bound to a single application*, but the future releases could include multi-application support without any impact on the security.

Concerning the design, the Authorization system of MORPHEMIC is an evolution of the MELODIC's one. MELODIC supports two parallel authorization functionalities:

- User Role-Based Authorization
- Infrastructure Pre-Authorization.

This *logical* separation is very useful, since the two functionalities have different scopes. The former regards the definition of the actions that groups of users are allowed to perform; the latter regards the privileges to access a certain Deployment Environment, which depend on internal and external factors, such as the *role* of the requester or the date-time or the geographical location of the infrastructure.

This logical separation in MELODIC is *technological* as well: this means that the two authorization functionalities are separated and use different technologies. In particular, the former is provided as part by the JWT Token based mechanism used for the Authentication system, which defines also a set of *roles* to be associated to users and the corresponding actions associated with these roles (RBAC), as shown in Figure 10.

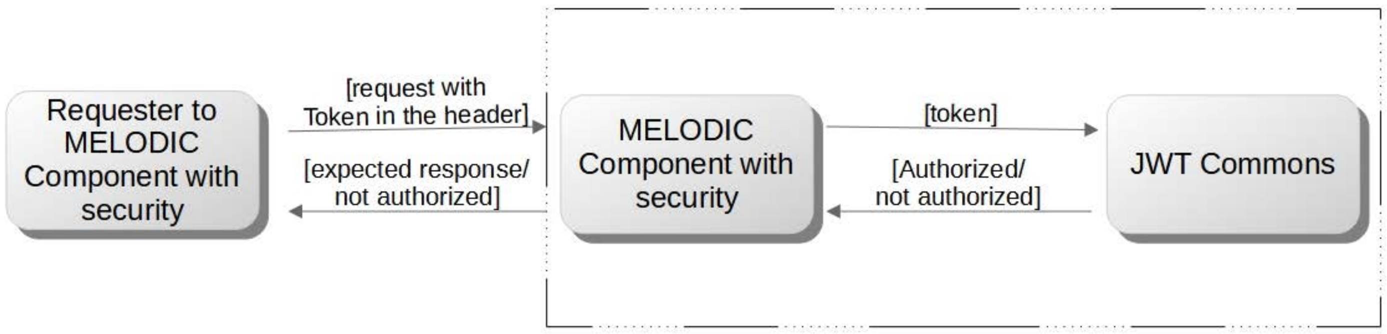
Requester to MELODIC Component with security
[request with Token in the header]
[expected response/ not authorized]
MELODIC Component with security
[token]
[Authorized/ not authorized]
JWT Commons

Figure 10 MELODIC's RBAC Authorization system

The Pre-Authorization, instead, is a dedicated system able to process and enforce complex policies based on *attributes* (ABAC).

It is easy to understand that the technological solution used to implement the latter functionality can easily implement the former. Exploiting this possibility will give the following advantages:

- simplification of the technological architecture: all the operations related to Authorization are implemented by using a single technology
- extensibility of the User-Authorization system, which can support complex attributes, not only roles

- complete separation between authentication and authorization, providing more manageability.

As a consequence, the MORPHEMIC Authorization system is based on an ABAC model, has been implemented through a XACML architecture and provides the following functionalities:

- User-Authorization, currently based only on user attributes, identified as *roles* to define user-related privileges
- Deployment Environment-Pre-Authorization, based on generic attributes, to define the privileges to access the Deployment Environments.

The next sections will detail the design and the technology of these two functionalities, providing the roles currently supported by MORPHEMIC and some examples of attributes.

3.2.1 User-Authorization and Deployment Environment-Pre-Authorization: concept and design

As mentioned above, the Authorization system of MORPHEMIC can be considered as composed of two logical mechanisms, derived from MELODIC: *User-Authorization* and *Deployment Environment-Pre-Authorization*.

Both the mechanisms implement the ABAC model, however the User-Authorization considers only the roles as attributes associated to the user, actually implementing a *Role-Based* model. In fact, a role represents the duty or the function that an *actor* has in a specific context and is strictly related to the functionalities available to each user: it can be seen as a particular kind of attribute of this user. According to this consideration, the User-Authorization in MORPHEMIC implements a RBAC model as a specification of an ABAC model.

The User-Authorization mechanism controls the user operations on the platform. Each MORPHEMIC user can be associated to the roles shown in Figure 11.

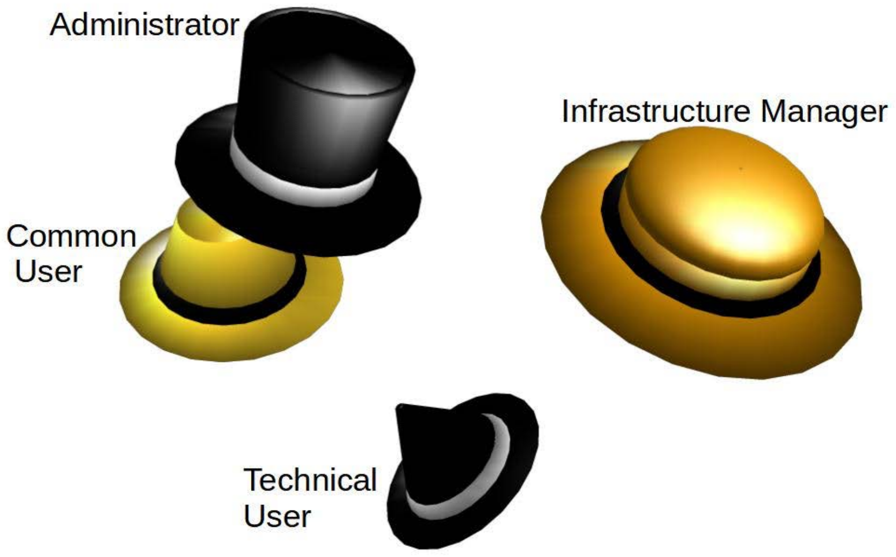

Figure 11 Roles in MORPHEMIC

This figure shows a series of hats, representing the roles supported by MORPHEMIC:

- *Common User*: can manage all the operations and resources related to a certain application, such as to perform deployment or manage associated Deployment Environments (but not associate or remove them from the availability of MORPHEMIC)
- *Administrator*: inherits the privileges of the Common User; furthermore, it can perform administrative tasks, such as user management (create user accounts, assign privileges etc.)

- *Technical User*: is used only internally by Metasolver component in order to create reconfiguration processes
- *Infrastructure Manager*: performs all the administrative operations concerning Deployment Environments, such as association, configuration, definition of the Pre-Authorization Policies and removal.

The first three roles are present also in MELODIC, with the same meanings and the same relationships: specifically, an Administrator is also a Common User and the Technical User is actually an internal role. The inclusion of the Infrastructure Manager is due to the extension of the range of supported Deployment Environments. The increased complexity of all the related management operations suggested the idea to define a specific role. Taking into account the hard-coded relationships among the roles, it is possible to combine some roles that can be obtained by users with some interesting privileges. For instance:

- Infrastructure Manager-Administrator, which has full control of the platform, including user management, infrastructure management and application management
- Infrastructure Manager-Common User, which has full control on the Deployment Environments and application related aspects.

The Pre-Authorization system is inherited from MELODIC, plays the same role and uses the same technology. Specifically, Pre-Authorization allows or denies certain users under certain conditions to access certain Deployment Environments. It is managed by the MORPHEMIC platform and is distinguished from the security mechanisms of the Deployment Environments. In particular, the involved components are the Adapter and the Scheduling Abstraction Layer, *before* the actual deployment starts (Figure 12).

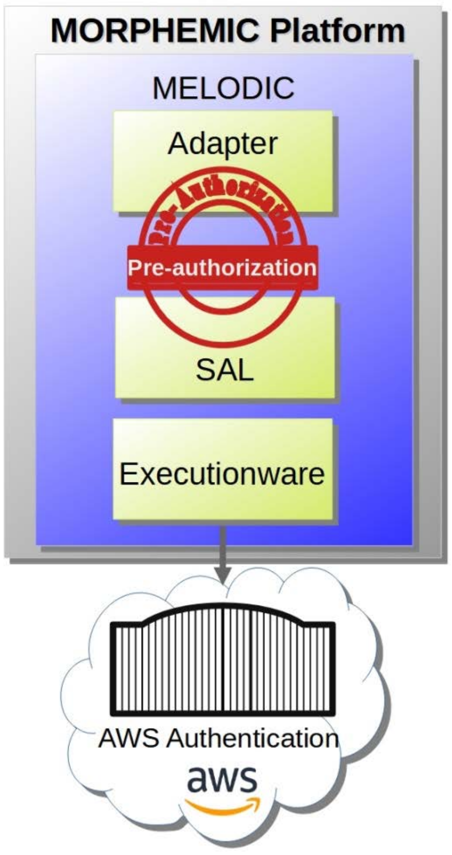

Figure 12 Pre-Authorization and environment's authentication processes

Pre-Authorization allows a fine-grained control on the accesses to certain Deployment Environments, limiting the deployment operation to certain users, certain periods, and certain locations. For example, it is possible to define policies that *deny the deployment of databases on Deployment Environments located outside the EU*. Similar policies could be defined also by the Deployment Environments, but in some cases, such as Edge Nodes or private Clouds, these aspects are not implemented such that a common mechanism provides an added value.

Pre-Authorization is a generic implementation of the Attribute Based Access Control (ABAC) model, since it considers a wide range of heterogeneous attributes, related to the user, the Deployment Environment (DE) or the context, which are retrieved in different ways. For example, DE attributes can be defined through configuration or can be retrieved by the Executionware; the model is very flexible and, if the future implementations of MORPHEMIC will support finer-grained user profile with several attributes, it is possible to use them to define complex policies.

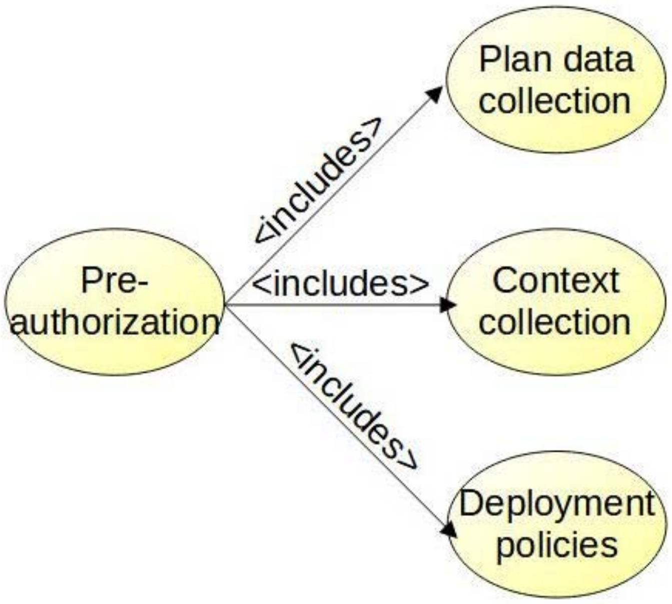

Figure 13 Use case diagram of Pre-Authorization

Figure 13 shows the use case diagram of the Pre-Authorization process:

- *Plan Data Collection* includes the retrieval of user and resource attributes by using different mechanisms
- *Context Collection* includes, for example, the retrieval of current date-time or current location
- The *Deployment Policies* are stored in a specific document associated to a certain Deployment Environment.

Examples of basic policies composed by the attributes used in the Pre-Authorization process are the following:

- "deny the access of certain *roles* to a certain *Deployment Environment* at *night-time"*
- "allow the access to a certain *Edge Node* only to a certain *user"*
- "limit *the list of locations* in which data can be stored".

These policies comprise attributes associated to the user, the environment and the resource:

- examples of user attributes are *username*, *role*, *source IP address, location of the user*
- *environment attributes* are *time*, *date*, *day of the week*
- *resource* (*Deployment Environment or node*) attributes are *location of the DE*, *type of DE*, *owner of the DE*, *number of cores of the node*, *memory*

The two logical functionalities, User-Authorization and Deployment Environment-Pre-Authorization are implemented on a unique XACML architecture based on the ABAC schema. As mentioned above, the RBAC model of User-Authorization is conceived as a particularization of ABAC. Policies in general are based on combinations of different categories of attributes.

Table 2 Examples of policies

Role based policy	*An Administrator is allowed to create users*
Attribute based policy	*A Common-User can deploy applications on infrastructures located in EU*
Attribute based policies with environment attributes	*A Common-User can deploy applications on infrastructure located in EU from 8 a.m. to 8 p.m.*

Table 2 shows three sample of attributes based policies. All the policies contain a *role* attribute taken from the roles listed in Figure 11: the first one is a pure role-based policy, the second considers also an attribute related to the infrastructure (location) and the third one also the environment (*time of the day*).

Another difference is that the first policy in MORPHEMIC is a *User-Authorization* policy, which defines the behaviour of the platform for the internal operations, the others are Deployment-Environment-Pre-Authorization policies, which should be defined by the *Infrastructure Manager* to control accesses to a specific infrastructure or group of infrastructures.

The first policy can be processed by using the same technology of the others removing the attributes not related to the user. This means that the functionalities of the specific RBAC system (Figure 10) of MELODIC can be provided by the ABAC system. However, the latter, whose architecture will be described later, is much more complex and introduces a degradation on the performance. In order to reduce this problem, preventing an impact on the overall performance of the platform, a careful evaluation of the components that need specific authorization controls has been conducted. The evaluation is finalized to provide a trade-off between performance and security (in terms of authorization). A set of components representing the interfaces for user-related functionalities has been defined: these interfaces should assure that the related functionalities are used only by authorized users. This reduces dramatically the number of components that should be integrated with the ABAC Authorization system, reducing also the performance degradation and preserving the simple and consistent approach defined.

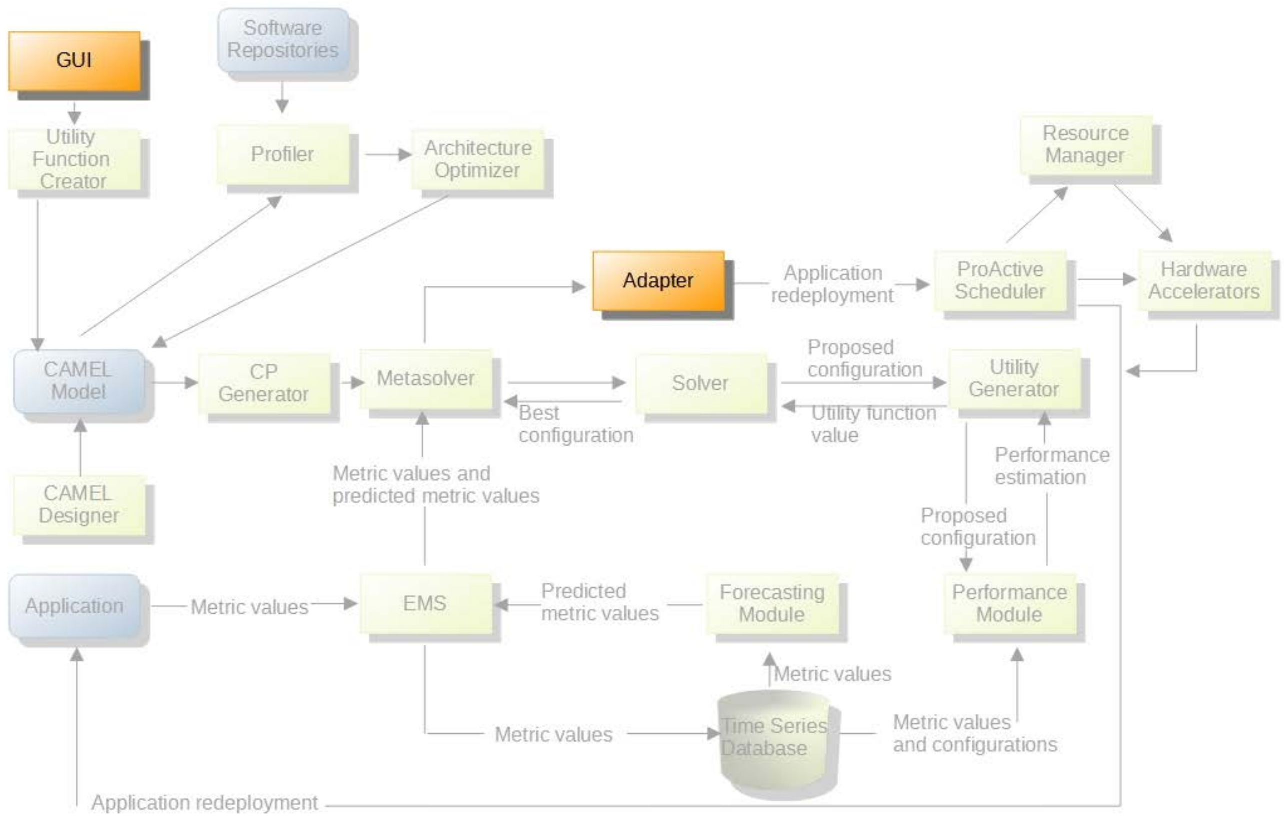

Figure 14 Components which need authorization

Figure 14 highlights, among the components of Figure 4, the ones integrated with the Authorization system. In particular:

- The *GUI* represents the main access point of the platform. It exposes the main commands for the user and potentially should be able to reconfigure also its presentation according to the allowed operations
- The *Adapter* is responsible for preparing a complete plan of application reconfiguration for new deployments (Deliverable 4.1 - *Architecture of pre-processor and proactive reconfiguration)*

At the time of writing this Deliverable, the Authorization system protects the aforementioned modules taking decisions based on the user attributes (roles and generic attributes), resource attributes (e.g., information related to the Deployment Environment, such as location) and environment attributes (such as date and time). Even if the architecture is completed and consolidated, it is possible that in the future new functionalities will be added. Furthermore, even during the development phase, or after the project, it is possible that the effectiveness of the Authorization process should be improved by integrating other components. The architecture of the XACML Authorization system can be easily integrated in service-oriented architectures.

The next section analyses how attributes are obtained, decisions are taken and policies are enforced.

3.2.2 Implementation details: XACML, AOP and interceptors

The implementation of the ABAC/RBAC Authorization system in MORPHEMIC is based on the *eXtensible Access Control Markup Language* (*XACML*). XACML is a standard produced by the *Organization for the Advancement of Structured Information Standards*[9] (*OASIS*) to represent access control policies in XML format. The most recent version of XACML is 3.0[10].

The standard is based on the following concepts:

- *Rule*, which defines an *effect* (decision, i.e., permit or deny) based on a *target* (defined by certain *attributes*) that can be the requester (*subject*) or the *resource*
- *Policy*, a set of relevant rules, whose effects (i.e., partial decisions) are reconciled using a rule-combining algorithm that produces the policy decision combination of rules based on a *rule-combining-algorithm*
- *Policy-set*, a combination of policies based on a *policy-combining-algorithm*.

Target is a type of pre-filtering condition that narrows applicability of a rule/policy/policy-set to certain subjects or resources or actions or combinations of them.

In technical terms, Target is checked using only the info already available from the authorization requests. ABAC engine does not collect info from other components. Rules/Policies/Policy Sets whose Target does not match the given authorization request info are simply ignored. ABAC engines evaluate Targets in order to quickly identify the relevant rules and policies and ignore the rest, thus improving performance and response time.

After pre-filtering with Target, an ABAC engine will evaluate the Rule conditions, and if necessary, it will contact other components (using plugins called PIP context handlers, Figure 15).

The other elements of an ABAC statement are the *Subject*, associated to who asks for accessing a certain resource and the *Resource,* associated to the resource to be accessed. Furthermore, the *Action* defines the action to which the decision is referred while the *Environment* defines the *context* in which a certain decision should be taken, such as a certain date and a certain time.

When a rule condition evaluates to *True* the corresponding Rule effect (permit or deny) is yielded as a partial authorization decision. If more than one rules in a policy yield partial decisions, then the Policy's rule-combining-algorithm is used to reconcile any decision conflicts and produce the policy decision. The same happens if two or more policies in the same policy set yield policy decisions.

Other concepts of XACML that are useful in MORPHEMIC are:

- *Advice*, an action that *could* be performed in case of a specific decision has been taken

[9] https://www.oasis-open.org
[10] http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

- *Obligation*, an action that *must* be performed in case of a specific decision has been taken.

In other terms, if a policy or a rule defines that a certain data set must be accessed by a *common user* with the *obligation* to notify an *administrator*, the Authorization system *must* notify an administrator and if it cannot, the data cannot be accessed. If the same policy contains instead the *advice* to notify an *administrator*, the data is accessed even if it is not possible to notify an administrator. Section 3.3 will show how these concepts are used in MORPHEMIC.

The architecture of a generic XACML-based ABAC Authorization system is based on the following elements:

- *Policy Administration Point* (*PAP*): the *point* which collects and administrates the policies
- *Policy Decision Point* (*PDP*): it receives authorization requests and makes the authorization decisions
- *Policy Enforcement Point* (*PEP*): it sends authorization requests to PDP and executes its decisions
- *Policy Information Point* (*PIP*): it provides complementary attributes to the PDP (e.g., part of subject or resources attributes and environmental attributes).

Figure 15 shows the generic architecture of a complete XACML-based ABAC Authorization system.

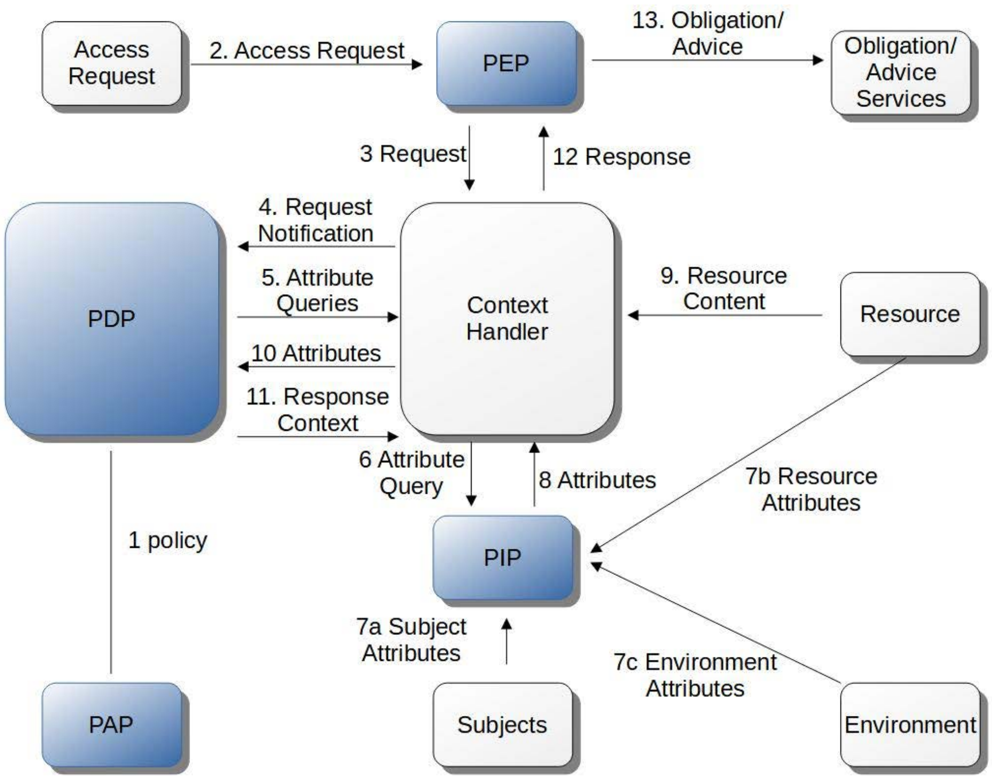

Figure 15 XACML generic architecture

The *Policy Information Point* gets the attributes from *subject*, *resource* and *environment* and provides them to the Context Handler. The Context Handler is the core of the architecture, since it gets all the needed information, including the details on the resource, such as the data stored, as well as requests and obtains the *decision* from the Policy Decision Point. The PDP, according to the policies stored in the PAP and the attributes obtained from the Context Handler takes a decision and communicates it to the Context Handler. Finally, the PEP enforces the decision and executes obligations or advices if needed.

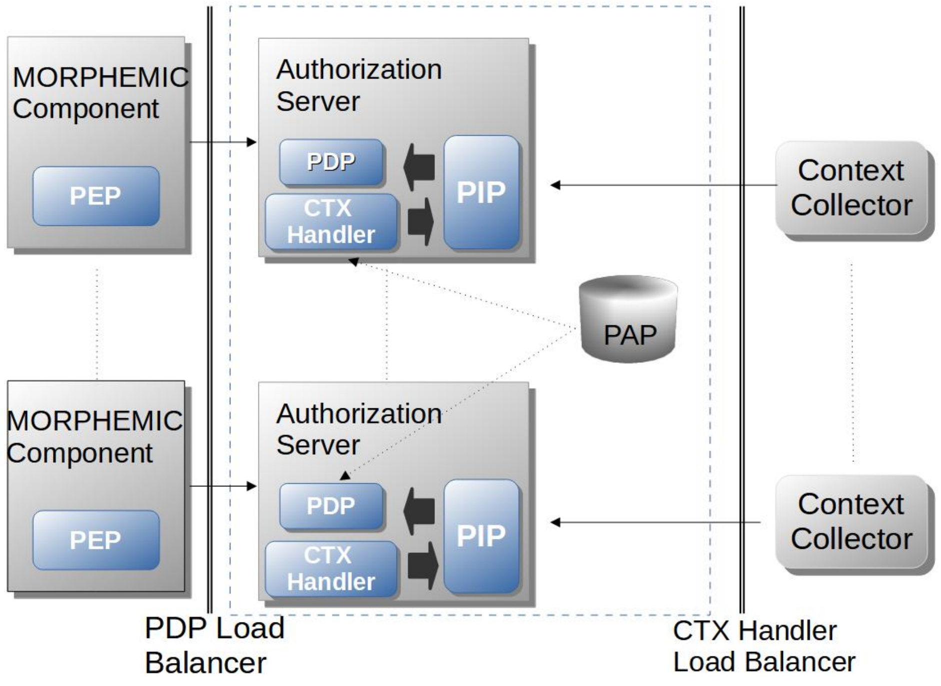

Figure 16 Authorization system in MORPHEMIC

Figure 16 shows how the architecture exposed in Figure 15 is deployed in the MORPHEMIC platform. Each of the components highlighted in Figure 14 is integrated with a PEP enforcing the decision taken in the Authorization Server. The Authorization Server contains PDP, Context Handler and PIP to collect the information and process the decision. MORPHEMIC can contain several instances of Authorization Server to balance the load and increase the performance. The PIPs instances get data about internal resources and external context (Context Collector). Finally, the PAP acts as a policy storage and provides the policies on which PDPs will decide.

The technological implementation of the MELODIC's ABAC Authorization system is inherited from MELODIC and is based on the XACML standard. It is built by using Balana[11], which is the implementation of XACML standard provided by WSO2[12] based on SUN's XACML Implementation[13]. As mentioned above, this technology in MELODIC implements only the Pre-Authorization system, while in MORPHEMIC also the User-Authorization system.

As MORPHEMIC uses internally MELODIC, a technological compatibility is considered very important. For this reason, the integration mechanisms used in MELODIC for authentication and authorization play an important role also in MORPHEMIC. Specifically, both aspect-oriented programming[14] (AOP) and Spring Interceptors[15] methods will be used in MORPHEMIC. They constitute different mechanisms to intercept methods calls and include pieces of code to be executed *before* or *after* the method invocation. The pieces of code can be used to decode the *token* and call the *authorization service*.

3.3 Access Logging system

The MORPHEMIC platform provides a functionality to log all the information about authentication and authorization attempts. This feature is implemented by the *Access Logging system*. The access information produced by the functionalities described in Sections 3.1 and 3.2 are stored as a log in a *tracker*.

[11] https://github.com/wso2/balana
[12] https://wso2.com/
[13] http://sunxacml.sourceforge.net/
[14] https://docs.spring.io/spring-framework/docs/2.5.x/reference/aop.html
[15] https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/HandlerInterceptor.html

Figure 17 shows the generic flow of the logging operation following a User-Authentication (Section 3.1.1) operation: the generic *Login Client* represents the User Interface (Figure 4), which, at the time of writing this deliverable, is the only user access point. The Authentication Server, which verifies username/password through the LDAP before issuing the *token*, forwards the *User-Authentication Log* to the *Access Logging system*.

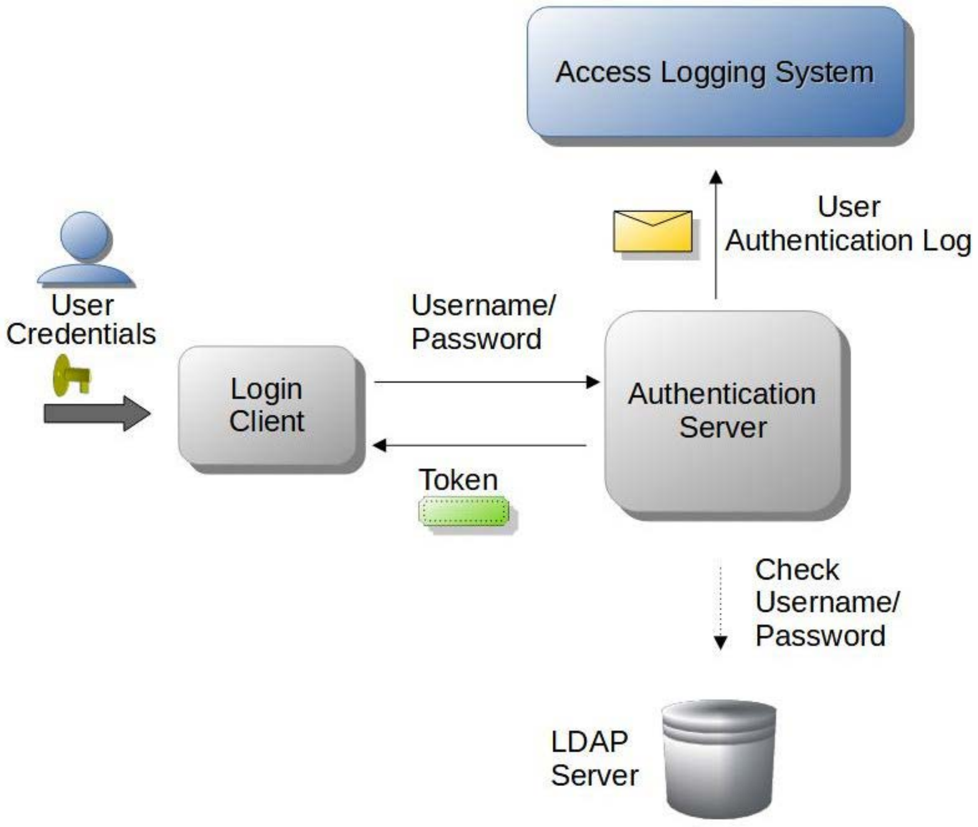

Figure 17 User-Authentication Log Flow

Figure 18 shows the same process in case of the *Deployment Environment Authentication* (Section 3.1.2): in this case the client which interacts with the infrastructure is the *Executionware* (represented by ProActive). It executes the operations defined by the Upperware components: specifically access the Deployment Environments through the stored credentials. The details and results of each access attempt are forwarded in form of *Deployment Environment Authentication Log*.

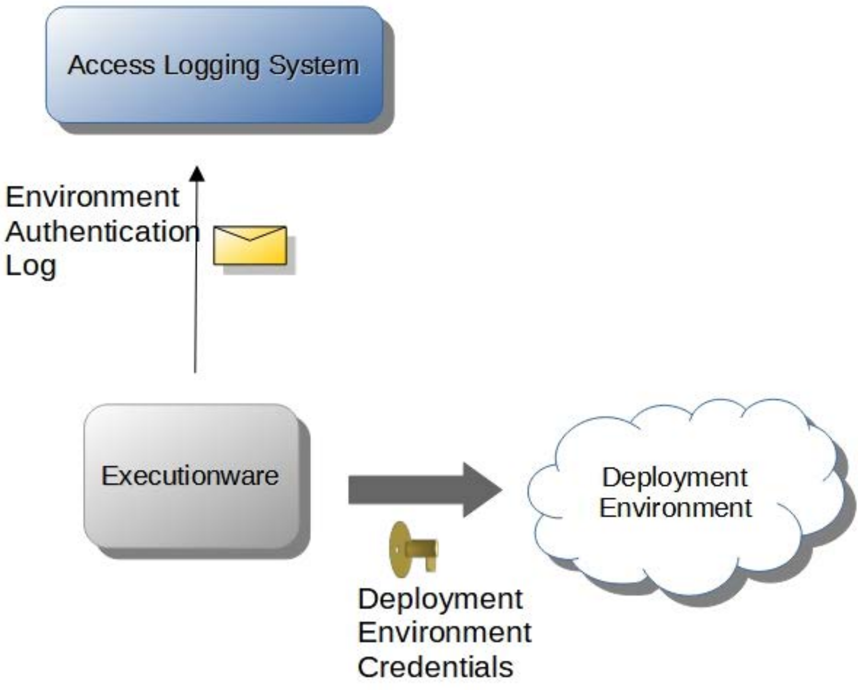
Access Logging System
Environment Authentication Log
Executionware
Deployment Environment
Deployment Environment Credentials

Figure 18 Deployment Environment Authentication Log Flow

Finally, Figure 19 shows the flow in case of Authorization (Section 3.2): in this case the Authorization Server forwards the *Authorization Log*. Specifically, the XACM standard include advices (Section 3.2.1) to ask for performing specific actions according to the result of an authorization decision: in this case, the advice requires to forward the Authorization Log.

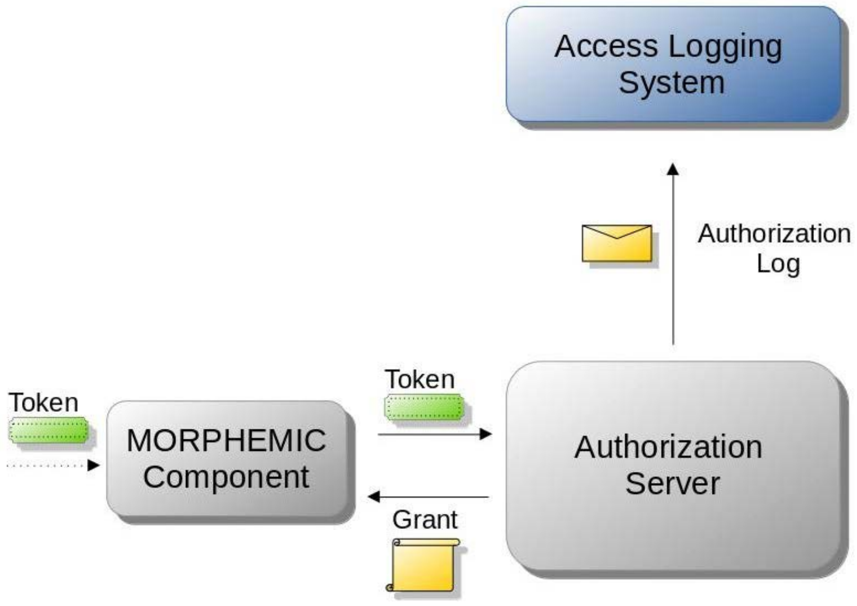

Figure 19 Authorization Log Flow

3.3.1 Architecture of the Access Logging system

The Access Logging system is *agent-based* and is integrated in the Authentication and Authorization systems. Figure 20 shows the integrated architecture, where in green the components of the Access Logging system are highlighted.

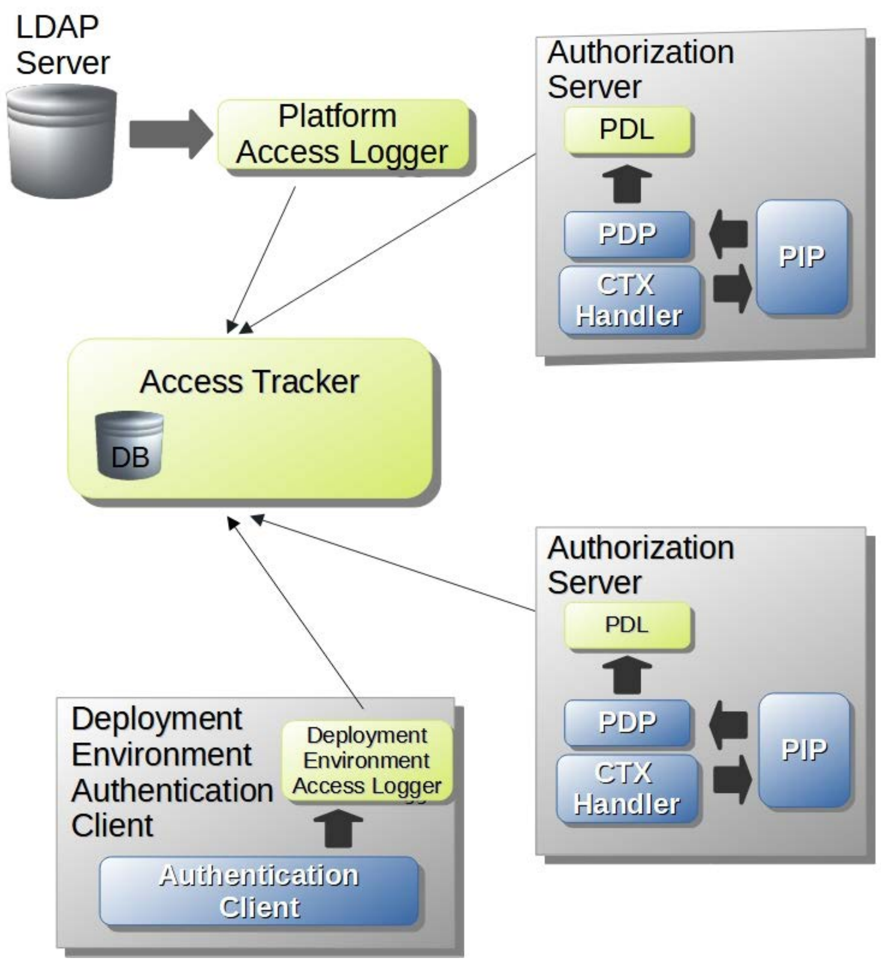

Figure 20 Architecture of the Access Logging system

The three agents are related to the three flows described in the previous section, specifically:

- *Platform Access Logger*, which collects and forwards the logs related to the User-Authentication
- *Deployment Environment Access Logger*, which collects and forwards the logs related to the Deployment Environment Authentication

Policy Decision Logger (PDL)*, which collects and forwards the logs related to Authorization. The PDLs are associated to the Authorization Servers (N load balanced instances,*

- Figure 16). The logs are forwarded to the *Access Tracker* which indexes them and enables easy access through REST APIs.

The log messages are formatted in JSON and contain the following fields:

```
Timestamp (mandatory)

User (mandatory)
```

```
Role

Action (mandatory)

Resource

      name

      action

Result (mandatory)

Info
```

whose meaning is the following:

- *Timestamp* the timestamp of the related event
- *User*: the username of the associated user
- *Role*: the associated role, if existing
- *Action*: defines the requested action, such as User or Deployment Environment Authentication or Authorization
- *Resource*: fields related to the resource (valid only for authorization)
  - *name*: the requested service/resource/environment
  - *action*: the operation requested on the resource
- *Result*: if the action resulted in a granted or denied request
- *Info*: other generic information, such as the error log or reason in case of denied authentication/authorization (for example, *obligation not executed*).

The mandatory fields (timestamp, user, action and result) are included in any log record. The other fields depend on the logged operation. Specifically, it does not make sense to include a role name for a User-Authentication action for some reasons:

- an unsuccessful authentication does not involve any role assignment
- an authenticated user could not have any role
- the role association happens *after* the authentication process.

The Resource Name and Resource Action are referred only to authorization operations, since authorization involves resource access. Finally, the Info field contains generic, optional information: in particular it is useful when the access is denied to specify the reason, such as an *internal error*, an *obligation not executed*, or an *invalid policy* etc.

3.3.2 Implementation details: Access Tracker and agents

At the time of writing this Deliverable, the technological architecture is completely defined, but the implementation has not started yet.

The Access tracker will be based on *Elasticsearch*[16].

[16] https://www.elastic.co/elasticsearch/

Elasticsearch is a search engine based on the Lucene[17] library: it provides a distributed, multitenant-capable, full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is dual-licensed under the source-available Server-Side Public License[18] and the Elastic License[19].

As shown in Figure 20, Elasticsearch (Access tracker) will get data from the agents through its REST API. Elasticsearch also provides API to search and retrieve them.

The agents will get the logs from the specific security components and forward them to the Access tracker. Their specific implementation depends on the associated flow:

- the User-Authentication Agent will be integrated in the LDAP to intercept the authentication requests (usernames) and responses
- the Deployment Environment Access Logger will be part of the Executionware which then communicates the cloud provider credentials to the ProActive server through the Resource Manager endpoints.
- the Policy Decision Logger will be integrated to the ABAC authorization server of MORPHEMIC in order to record authorization requests, the (final) authorization decision, any partial decisions (per policy or policy rule), as well as the attributes and their values, used to evaluate a request against the active ABAC policies.

Table 3 specifies for the aforementioned cases, which fields of the log message are used along with the mandatory ones.

Table 3 Fields used in the MORPHEMIC's security logging messages

Case (Agent)	Fields
User Authentication (User-Authentication Agent)	Info (in case of authentication failed to specify the reason or error)
Deployment Environment Authentication (DE Access Logger)	Resource.name (used infrastructure) Info (credential used – password/secret keys not logged)
Authorization (Policy Decision Logger)	For each decision (partial and final): Role (user role) Resource.name (service or infrastructure) Resource.action (action requested) Info (if a reason message is associated to a denied authorization it is logged here)

[17] https://lucene.apache.org/
[18] https://www.mongodb.com/licensing/server-side-public-license
[19] https://www.elastic.co/licensing/elastic-license

4 Deployment Security

In MORPHEMIC, the Deployment Security is defined as *the capability to select the most suitable Deployment Environment according to the security requirements of the application*.

This research topic is not consolidated yet, but some work is being done. For example, the *Cloud Security Alliance*[20] defined a matrix to control the security related features provided by Cloud infrastructures[21]. Other EU projects are focused on specific certifications to define an objective *security level* of cloud infrastructures (MEDINA[22]).

This is not the main scope of MORPHEMIC, however, some use-cases require a certain level of control on the security features when a Deployment Environment is selected, on the other hand MORPHEMIC should not limit the analysis to the Cloud but embrace also the other supported Deployment Environments. This chapter provides a minimal analysis and a potential solution that could be improved during or after the project.

4.1 Introduction and motivations

Most of the requirements listed in Table 1 refer to this aspect, along with several implicit requirements related to the capability to support specific Deployment Environments.

Table 4 Requirements indirectly related to deployment security

ID	Description	Category
MOR-SE.2	MORPHEMIC must support Hybrid Clouds	Supported Environments
MOR-SE.6	MORPHEMIC must support on-premises environment	Supported Environments
MOR-SE.10	MORPHEMIC could support environments different than the ones listed in MOR-SE.1-MOR-SE.9	Supported Environments
UC-C-SE.1	MORPHEMIC must support hardware	Supported Environments
UC-C-SE.2	MORPHEMIC must support deployment configurations spanning across multiple geographical locations	Supported Environments
UC-C-UF.4	MORPHEMIC must support geographical constraints in the deployment configuration	Parameters for the Utility Function

Table 4 shows some requirements related to the supported environments. For example, some use cases ask to control the *geographical location* (UC-C-SE.2) for data storage, usually because it is important for GDPR; in other cases, the support for on-premises environments (MOR-SE.6) or hybrid clouds (MOR-SE.2) are requested to have much more control on the physical location where data is stored. These are all motivations related to security and, in most cases, the motivations behind the requirements impacting on the Deployment Environments are related to security.

[20] https://cloudsecurityalliance.org/
[21] https://cloudsecurityalliance.org/research/cloud-controls-matrix/
[22] https://cordis.europa.eu/project/id/952633

The identification of a series of constraints defining a limited set of usable Deployment Environments or nodes is the simplest solution to assure an acceptable security level to specific applications. For instance, considering a database hosting personal data, it is possible to ask MORPHEMIC to use an *on-premises* node to host it.

Some limited options on this sense are already available by leveraging the mechanisms provided by the Platform Security. Specifically, Pre-Authorization is very flexible and, potentially, can define policies related to geographical location or other resource attributes to help to increase the security of the overall deployment. However, the main limitation of this possibility is due by the fact that the policy is applied *after* the selection of the Deployment Environment or Node and does not include an alternative. In other terms it is possible only to decide if the database can be deployment on that specific node, not to determine the best node where the database can be deployed.

According to this consideration, to build a decision mechanism about the most secure Deployment Environment or Node basing only on the Pre-Authorization is not specific and, even if it was integrated to cover the missing aspect, probably it would be cumbersome.

A more interesting option is to identify the *candidate* Deployment Environments and *candidate* nodes, according to a set of security parameters and select them only if some security requirements related to the *application* are met. Specifically, the CAMEL model includes information on the needed components, that, provide an idea on the security requirement, as well as the deployment model.

At the Deployment Environment side, the definition of an associated *security profile*, implicitly extended to the nodes, is needed. The security profile is a set of information defining the security parameters provided by the Deployment Environment. These parameters are listed and detailed in Section 4.2: at the time of writing this Deliverable, it is almost impossible to retrieve them from the Deployment Environments through queries or any sort of automatic inspection. The most effective solution seems to be the manual definition of a *default security profile* associated to a certain Deployment Environment to be used for reference when a certain security requirement is raised by a certain application. In particular, especially for on-premises and edge environments, the Infrastructure Manager (Section 3.2.1) manually defines the *security profile* of the Node or Deployment Environment (Section 4.2).

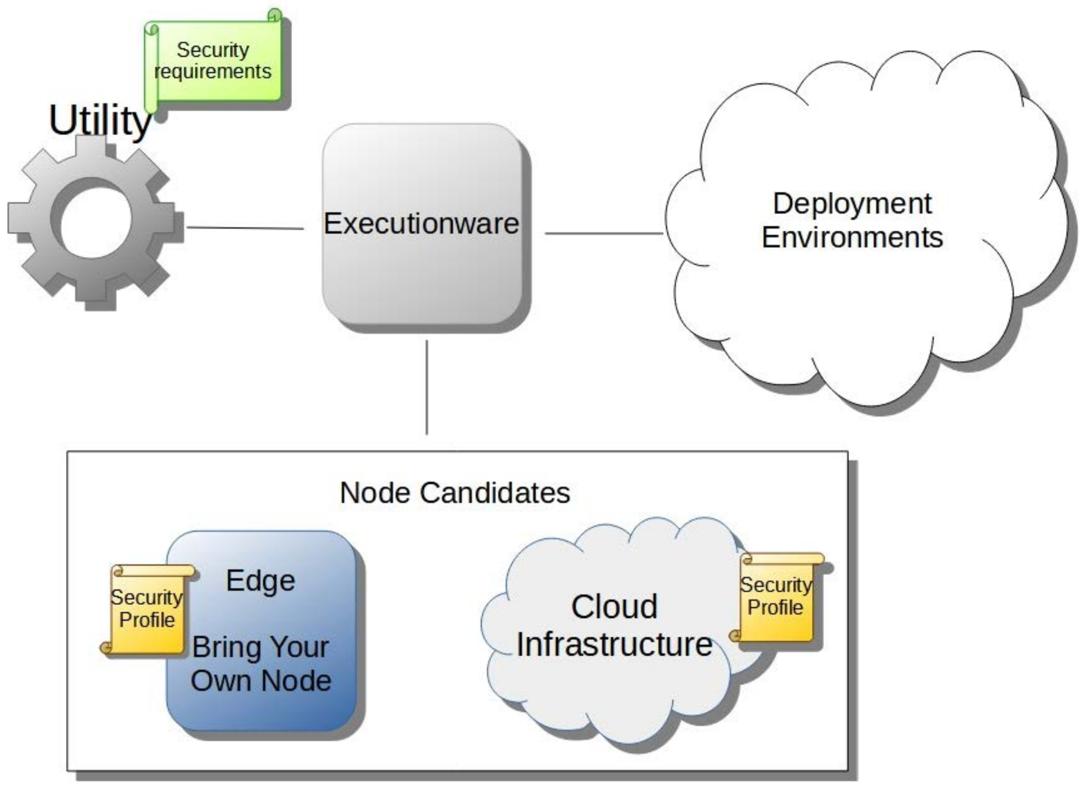

Figure 21 Deployment security flow

Figure 21 shows the flow of the Deployment Security system of MORPHEMIC. The security profiles are associated to specific groups of Node Candidates (in this case, belonging to a Cloud infrastructure and a Edge group of nodes) at the Executionware level. The security profile integrates the *infrastructure profile* needed to associate the Deployment Environment to the platform. In order to take a decision on the selection of the Node Candidate for the deployment, the security profile is matched against the *security requirements* in order to maximize the Utility that will take into account also the security parameters (*Security Utility*).

The Deployment Security integrates the core functionalities of MORPHEMIC; in other terms, it provides new parameters on which the selection of deployment nodes will be performed. In this sense, there is not any specific component of Deployment Security, but some components, involved in the evaluation of the Utility and in the selection of the Nodes, are impacted to implement this functionality. Specifically, the *Utility Generator* should take into account the security utility values, while the Executionware (Section 4.3) will host the data related to the security profile of the Deployment Environments.

In particular, as suggested in Figure 22, the Utility Generator, CP Generator, Solvers and Metasolver cooperate to select the best Deployment Environment offer according to the security parameters provided. These components will produce a deployment instance model which the Executionware will deploy on the selected Deployment Environments.

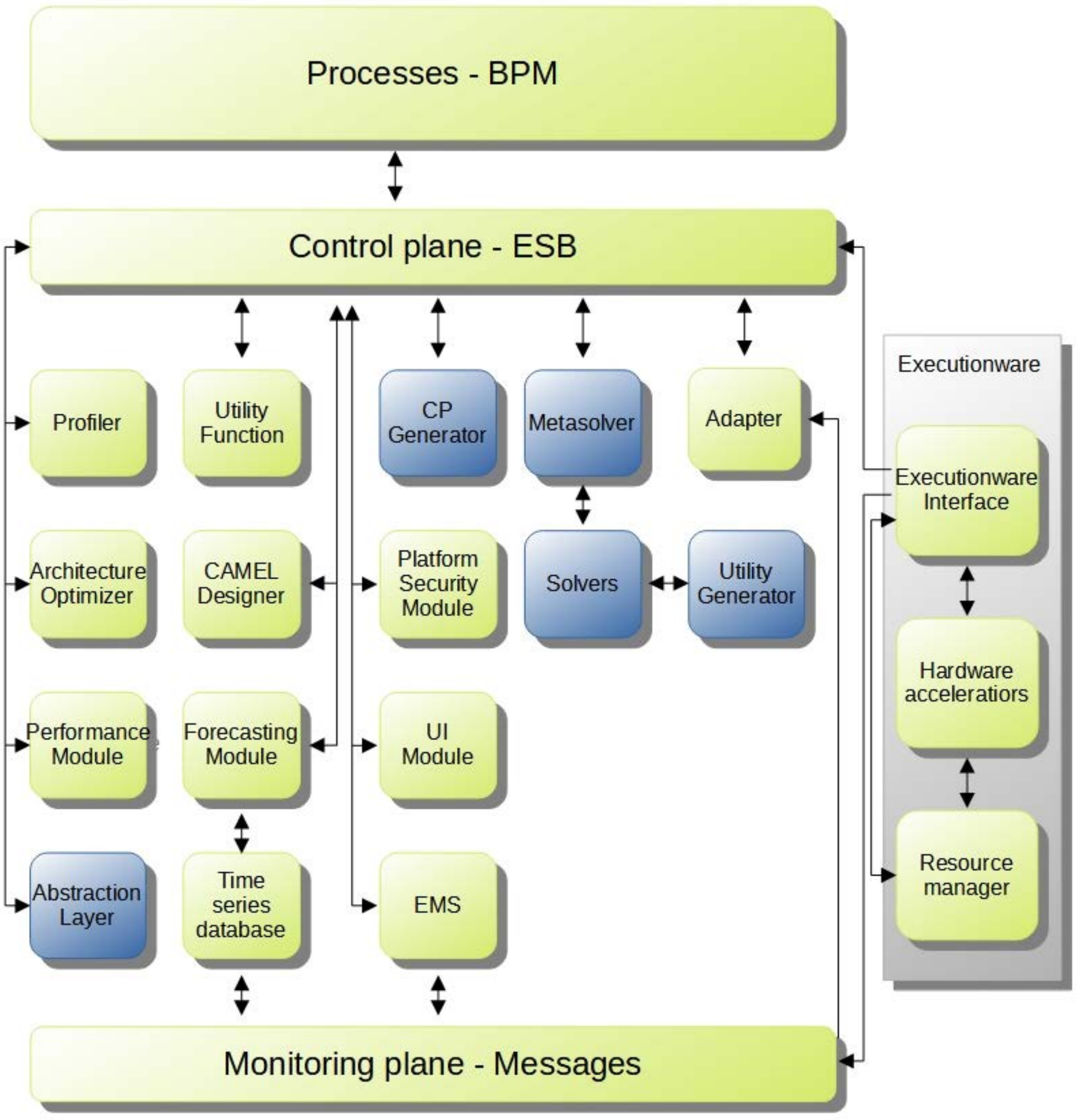

Figure 22 Deployment security in the MORPHEMIC architecture

The Executionware is also involved in the process, as it stores the *security profile* along with the other information associated to Deployment Environments, in order to present the Node Candidates with the related security parameters.

The security requirements of the application are related to the security requirements if its components. For instance, storages, web interfaces or service buses require certain specific security features from the node on which they are deployed. In other terms, the *security features* requested to a certain node are related only to the element (component or service) to be deployed on it and cannot be manually defined by the user. This simplifies the configuration and prevents the user from getting improper security levels for specific components, reducing the possibility to have insecure configurations.

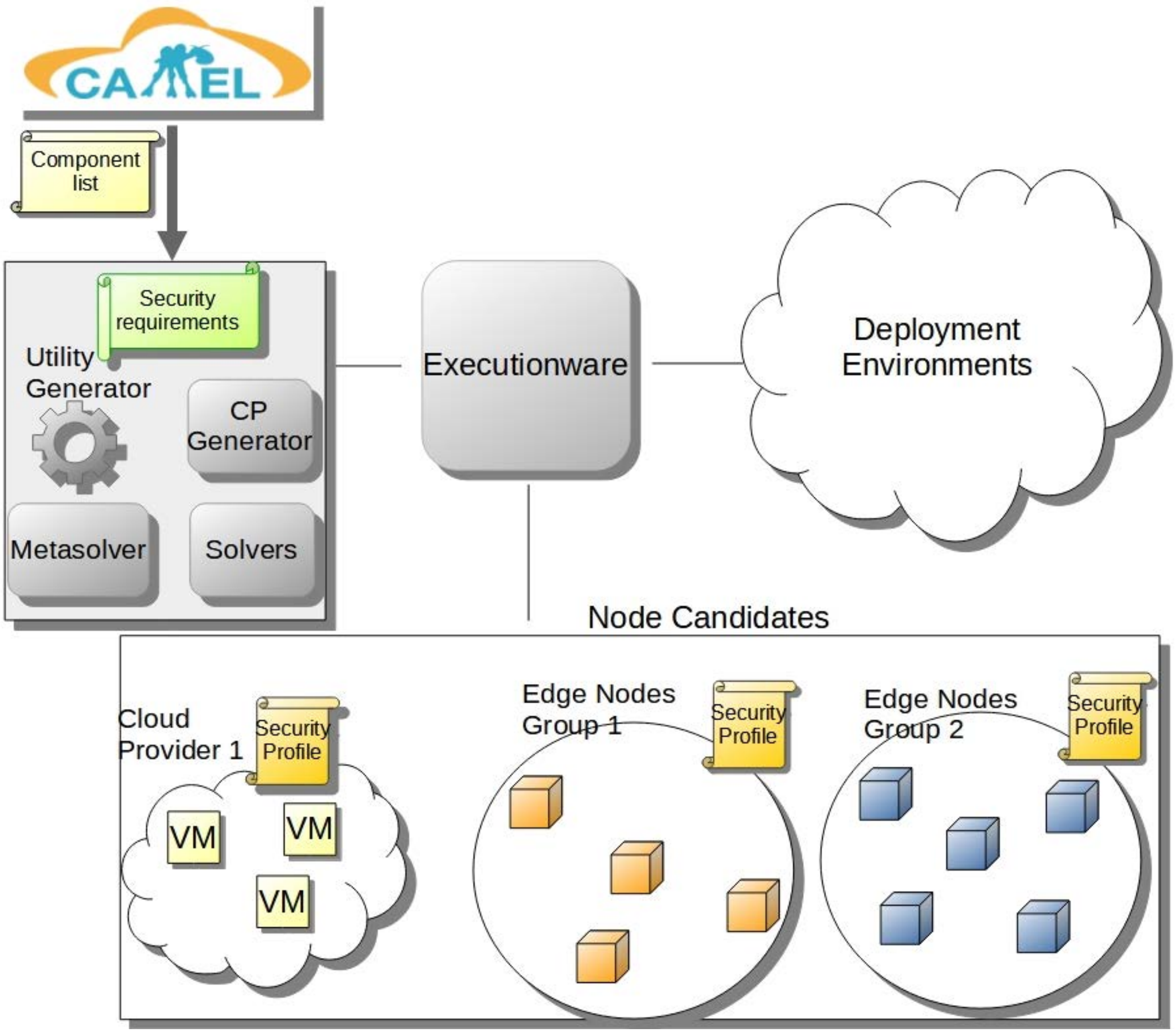

Figure 23 Deployment Security's detailed flow

Figure 23 shows the flow of Figure 21 detailed to better expose the previous considerations. Specifically, the CAMEL Model defines the *role* of each component to be deployed (database, server etc..) on a certain node: this can be interpreted as a *component list* from which the *security requirements* can be retrieved for each node. In particular, in CAMEL, the component definitions give information of the functionalities of each component through a specific annotation from the Meta Data Schema (MDS), and, as a consequence, these functionalities imply the security requirements of their hosting nodes.

The Executionware retrieves a list of node candidates according to the CAMEL model: the list is stored in a database. The *Solver* takes the final decision in order to maximize the utility taking into account also security parameters and forwards the decision to the Executionware for applying it.

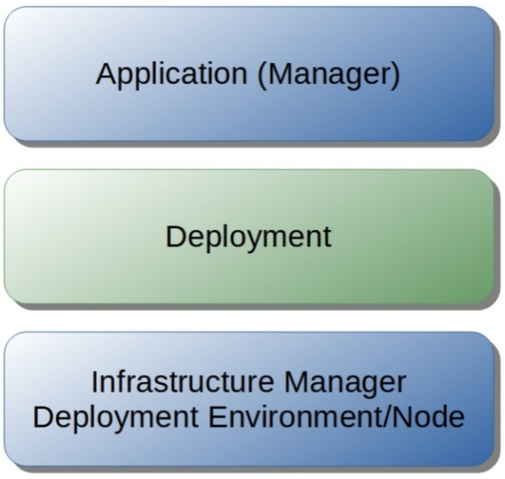

Figure 24 Security responsibility levels

The flow described above, shows that MORPHEMIC introduces a new responsibility level, in terms of security. Specifically, now, the security level of an application depends also on the deployment configuration. Figure 24 shows the introduction of an intermediate responsibility level, related to MORPHEMIC. In particular:

- in legacy application deployment procedures, the *Application Manager* is responsible for producing adequate security requirements. By using MORPHEMIC, he or she is mainly responsible to produce the right CAMEL model, defining the needed components with the required detail level. The *Application* by itself must be developed and deployed according to the best practices in terms of security
- as responsible to the configuration of each new Deployment Environment, the *Infrastructure Manager* must assure that the security features of the *associated* Deployment Environment and its Nodes are correctly mapped. More in general, the Deployment Environment and the Nodes must not present security holes and should in terms of network or platform (e.g. latest security patches installed)
- the *Deployment* level is the main responsibility of MORPHEMIC, in particular of MELODIC, considered as one of its components: it includes matching the *security requirements* of each application component, with the *security features* of the nodes. The application *security requirements and* the Deployment Environment *security features* are expressed in terms of *security parameters* (Section 4.2).

The following sections will detail the main components involved in this flow:

- defining the functionalities and the technical aspects of the *Deployment Environment security parameters* (Section 4.2)
- how the Scheduling Abstraction Layer, supported by the ProActive Scheduler, retrieves the candidate nodes (Section 4.3)
- how the *profiling, reasoning and adaptation tasks* evaluate and use the security parameters and the security requirements so as to select the best nodes (Section 4.4).

4.2 Deployment Environment security profile, security requirements and security parameters

Deliverable 5.3 (*Deployment artefact manager*) describes how the Executionware manages the association of a new infrastructure through the method *addCloud*. Specifically, this method defines a Deployment Environment through a set of parameters like the *endpoint*, the *infrastructure type* etc. This *profile* is stored by the Executionware that identifies each node candidate according to the profile of the related Deployment Environment. Concerning security, the concept is very similar: a set of *security parameters*, associated to the Deployment Environment, integrates the mentioned profile, defining a *security profile*. This profile is used by the Upperware to select the best node candidates for a specific deployment.

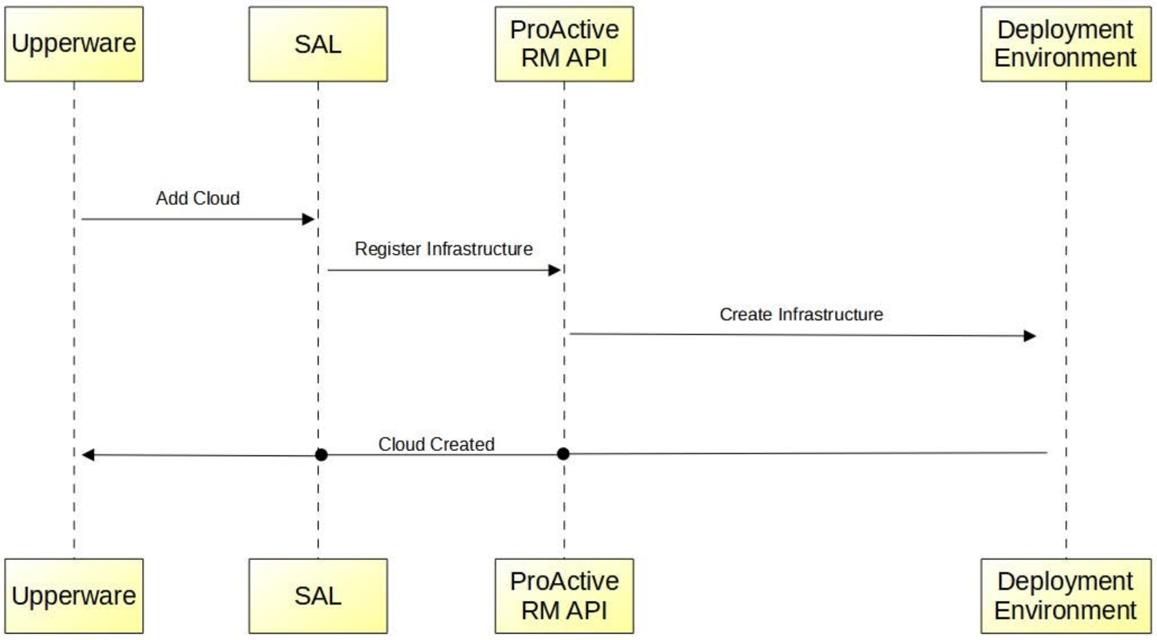

Figure 25 Infrastructure registration flow

Figure 25 shows the flow of an infrastructure registration (details on Deliverable 5.3). The registration process is started by the operation *addCloud*, which is associated to a set of data (Json format), including the infrastructure(s) endpoint(s) and credentials: part of this flow has been already described in Section 3.1.2, concerning the Deployment Environment credential management (Figure 9). The security profiles of the Deployment Environments will be included among this set of data and stored by the Executionware to identify the Node Candidates associated to the specific Deployment Environment.

The *Deployment Environment security parameters* are the core of the *security profile.* They define the security level of a Deployment Environment according to its characteristics in terms of hardware, operating system, network, access control and security certifications. In general, the security profile is manually associated to the Deployment Environment. In other terms, each Deployment Environment, from a big Cloud Provider (e.g., AWS) to a small node provided directly by the user (BYON), must be associated to a specific security profile.

Security profiles are manually defined, since it is not possible to elicit it automatically from the Deployment Environment. However, even if for a generic node the configuration should be fully manual, for the most widespread Deployment Environments, especially cloud, default security profiles will be made available.

Some Deployment Environments allow to instantiate with specific security features; in this sense, the ProActive Scheduler can instantiate those security features along with the nodes at deployment time if requested. This aspect is exploited in MORPHEMIC by associating multiple security profiles, mapped on the *advanced* security features, to the same Deployment Environment, as described in Section 4.3.

The security profile is a characteristic of the Deployment Environment and is associated to all the Node Candidates coming from that deployment environment: it is composed by some *security parameters.* They have been selected to have a thorough assessment of the security of a deployment node, be it a private or public Cloud infrastructure or any other networked device or service, such as, for example, an IoT device. At this purpose, several aspects must be taken into consideration: from the physical characteristics of the hardware involved to the software the runs over it, up to the security practises adopted by the people managing it or even its physical location.

Table 5 Deployment Environment security parameters

Security Category	Parameters
Authentication, Authorization, Accounting	Authentication, Authorization, multi-tenancy/tenant segregation, Access Logging
Hardware Level Security	X86, AMD SEV/SME, Intel SGX, EFI, AR, ARM Trustzone, FPGA, Bitstream Encryption, BootCodeAutentication, IoT
Network Level Security	Network segregation (VLAN), IDS/IPS, Firewall/NAT, Reverse Proxy, https everywhere, external/internal Vulnerability Assessment
OS-level security	Cgroups, selinux, Spectre/Meltdown/(etc) patches, general sw hardening, minimal OS images, event logging (+ centralized collection)
Physical access	Military grade premises vs self-hosted or publicly available
Security certification	CSA certs, ISO 27001/GPDR
Storage-level security	Data Encryption, Data Location

Looking at Table 5, the categories and parameters presented are used to answer several security related questions. Here is a rundown of those questions, divided by category:

- *Authentication, Authorization and Accounting* (*AAA*): do users need to provide any means of identification in order to interact with the infrastructure/service? Can they see or access objects or instances that are not their own? Can they freely perform any action, or are they limited in scope depending on their privileges? Is any action performed by users traced?
- *Hardware-level Security*: is the hardware involved exposed to know vulnerabilities? Can it be configured to boot only Operative Systems that have been certified to run on that specific SKU? Can it segregate virtual instances running on an hypervisor?
- *Network-level Security*: Is the incoming network traffic filtered in order to limit the attack surface? Is the incoming web traffic managed in a centralised fashion? Is it encrypted? Are there any measures to limit/prevent or misdirect any malicious traffic in place? If there are multiple internal networks, is the internal traffic separated by the use of physical or virtual segregation? On the public-facing networks, what are the results of a vulnerability assessment? Is it resilient to a multitude attacks that can debilitate the exposed services, such as Denial of Service?
- *OS-level Security*: If there is an Operative System running on the physical or virtual device, does it include any mean to avoid or contain privilege escalation? Can it segregate running processess? Does it exclude software that is not required for its purpose in order to minimise the attack surface? Does it include software patches for the most critical/well know vulnerabilities? Are user actions and software errors logged? Are the logs centralised?
- *Physical Access:* How easily can the hardware be accessed? Is it hosted inside safely guarded premises or freely accessible to the public? Security certifications, containing a list of the security certifications associated to the DE provider
- *Security certification:* can the service provider offer any certification that proves that the offered services are secure and/or private data is safely guarded and stored according to local/national/European laws?
- *Storage-level Security:* is the contained data freely accessible and readable or is it encrypted? Where is it physically located?

As it can be expected, the more positive are the answers to these questions, the better the security level of the infrastructure can be graded. However, it should be noted that, when evaluating the security grade of a deployment node, not every parameter enumerated in Table 5 might be applicable. For example, a network of Edge devices could offer no security certifications, performing a vulnerability assessment or a DDoS resilience evaluation of a Cloud provider infrastructure might be forbidden by local laws or require a previous agreement with the provider itself, etc.

Even if it is desirable to have always a Deployment Environment with the best security level, the overall *Utility* should depend also on other features matching the application requirements. An adequate solution in term of security can be obtained by comparing the security features of the Deployment Environment with the *security requirements* of the application. The security parameters shown in this section are used to express both: the matching process is shown in Section 4.4.

4.3 Executionware: definition of the candidate nodes

The Executionware collects information about the available Node Candidates of the Deployment Environments. The information consists of the node regions, images, hardware, and security profiles. Once the information is collected, the Executionware will compile a list of available node candidates and store them in a Executionware Database.

The information regarding the node candidates is vital for the operation of the Upperware. Based on the list of available nodes and on the application requirements, the Upperware decides which nodes out of the node candidates to be chosen for the deployment of the application. As a result, the list of node candidates is provided to the Upperware using the Executionware database and without the need of contacting the underlying infrastructure.

One of the approaches to evaluate the placement of the application is the security profile. The security profile is specific to the infrastructure where the nodes are hosted and evaluated in terms of their security. For some Cloud providers, such as AWS, a default security profile is provided while extra features can be added on demand like encryption and network firewalls.

An example of the process is the following:

- an instance on AWS is created with security groups acting as a virtual firewall to control inbound and outbound traffic
- by default, if the security groups rules are not listed, AWS will create instances that allow inbound traffic from network interfaces (and their associated instances) that are assigned to the same security group, and outbound traffic to all addresses, ports, and protocols
- the instance's security can be boosted by using a white-list approach where only the ports and protocols used by the applications are exposed.

From this example and from similar other examples, it can be remarked that the definition of a single security profile representing the security level of the whole Deployment Environment is a limitation in many common use-cases. In general, the security requirements vary according to the deployed instances.

MORPHEMIC helps on this implementing a hybrid approach: node candidates are evaluated according to a per-environment security profile. However, during the deployment phase the ProActive's Connector-IaaS supports the extension with additional security features. Taking into account these features, it is possible build different security profiles associated to a single infrastructure: for example, a default security profile for the default deployment and an advanced security profile including the extra security features. In other terms, the Executionware apparently provides different Node Candidates with different security profiles but with the other features identical: at deployment time, if a non-default security profile is selected, the ProActive's Connector-IaaS will extend the node with the advanced security features.

4.4 Application security requirements and Deployment Environment security profile in MORPHEMIC's profiling, reasoning and adaptation tasks

One of the key objectives of MORPHEMIC is the ability to automate and optimise cross-cloud, polymorphic application deployments. All the MORPHEMIC's components use a common application model, based on CAMEL, which can be seen as a superset of TOSCA[23] standard of OASIS. The model itself is Cloud agnostic and not dedicated to any particular

[23] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

cloud provider. The resulting application model is translated to a *CP Model*, providing a mathematical deployment optimisation model for the application, also in a cloud agnostic way. The security level of a *Candidate Node* is defined by its security profile, composed by the *security parameters* described in Section 4.2: they are, in most cases, inherited from the infrastructure (security) profile. The *application security requirements*, instead, are the security features requested by the application components and services to obtain an adequate deployment security level. The Node or Deployment Environment security profile defines the environment's constraint to be matched with the application security requirements retrieved from the CAMEL model and its included utility function and then translated to CP Models.

The pieces of information mentioned above are used in different tasks of the MORPHEMIC's processing flow. In particular, in the functionalities provided by MELODIC, i.e. the profiling, the reasoning, the adaptation and the deployment.

The CAMEL Model is handled by the *CP Generator* and converted into CP Model in order to be used in the following reasoning task. This first activity is performed by the *CAMEL* and *CP Generator* components:

the *CDO Client library* is responsible for parsing CAMEL Model to a corresponding in-memory object model. The security requirements will be available in the CAMEL model as annotated attributes of the respective entities (e.g., application components).

- the *CP Generator* based on the CAMEL model creates the *CP model*, used in the reasoning phase.It might encode security-related requirements using security variables, mapping to node candidate security parameters, for further reducing the solution space.

The reasoning task is the main functionality provided by the MELODIC component, since it is responsible for finding optimal deployment solutions for the given application utility within given constraints. The key components involved in the reasoning phase are:

- *Metasolver*, which chooses the right *Solver* for the specific deployment optimisation problem (I\i.e., the CP model)
- the list of *Solvers* is described in detail in deliverable 3.3 (*Optimization planning and adaptation approach*); *Solvers*, which will process the optimization problem that will include the application security requirements and the Deployment Environments security profiles among the parameters
- *Utility Generator*, which is responsible for the calculation of the utility values of the proposed solutions. The *Utility Generator* evaluates the Utility Function including security parameters. The Utility Function is defined in the CAMEL model.

The *adaptation task* deals with the adaptation and orchestration of the deployment model to be deployed to the chosen Deployment Environments. It is implemented in the *Adapter*, which prepares and orchestrates the deployment plans.

The *Executionware* is responsible for the deployment of the solution prepared by the Adapter to the chosen Deployment Environments. This includes assuring that after each reconfiguration the connections among the components are preserved.

The *monitoring task* collecting the measurements on the current execution context using an instance of Esper event processing[24] for gathering and aggregating the context metric values.

These latter tasks are focused on the deployment of the optimal solution found during the profiling and the reasoning task.

The described processing and information flows do not impact neither on the high-level architecture of MORPHEMIC, nor in the specific architecture of the MELODIC component. However, they require some modifications on the components implementing the profiling and reasoning task, along with the modelling (if necessary, mainly at the metadata/MDS level). The following components will be supporting security-based optimisation. However, they require modification on some of the components implementing profiling and reasoning task (e.g. the CP Generator), along with the modelling (if necessary, mainly at the metadata/MDS level):

[24] https://www.espertech.com/esper/

1. the CAMEL Model will provide information on the security requirements of the application (using suitable annotations coming from respective security-related elements of the MDS).
2. based on the CAMEL Model the CP Model for *reasoning* task is prepared, including security requirements and preferences in the constraints and utility function, respectively
3. the CP Model is enriched with actual values of the metrics
4. the most suitable *Solver* for given CP model is chosen
5. the *Solver* solves the problem; each solution is evaluated by the *Utility Generator* via the Utility Function which includes security requirements, as defined by user in step 1.
6. the best solution is passed to *Adapter* to be deployed.
7. the solution is deployed by Adapter using Executionware.
8. the monitoring task starts collecting the metrics of deployed solution
9. based on the metrics values the *Metasolver* could decide if the reconfiguration should be started
10. if reconfiguration is started then the *Solver* starts to look for the optimal solution, based on the new values of collected metrics; as in the previous steps, for each found solution, the utility function value is evaluated to define and compare the solutions; during the reconfiguration phase the security requirements will be handled in the same way as in the initial deployment.

The above process allows to include the security parameters of the candidate nodes in the deployment and reconfiguration processes of the MORPHEMIC pre-processor.

5 MORPHEMIC's reused components: security analysis

The security level of a platform is given by its security-related components, such as Authentication and Authorization systems, security services, such as IDSs, from the security features of its generic components and their vulnerabilities.

While the previous chapters focused on the Platform and Deployment Security, this chapter is dedicated to the vulnerabilities of the inherited components. Each component must be integrated to the Platform Security system, as described in Chapter 3, but this is not enough to limit the possibility of malicious attacks. It is possible that the generic components present vulnerabilities opening attack points: this issue should be faced at design time (security by design [5] [6]) or at development time. For the components inherited from other projects, this aspect is more critical, because it should be assured that they are up-to-date, concerning the security and that they do not present specific vulnerabilities. In particular for some EU projects, it is not guaranteed that the outcome present an adequate security level, and this should at least be analysed, and, possibly, fixed before the integration in MORPHEMIC. To this end, a vulnerability assessment is useful to discover and fix issues that could impact the sophisticated platform that will be the outcome of the project. The vulnerability assessment aims at attaining a system hardening scheme with the potential use of different techniques. In general, *hardening* is defined as the process of securing a system by reducing the *attack surface*, i.e., the sum of the attack points which can be used by unauthorized users (*attackers*) to enter or extract data from the system.

Several components of MORPHEMIC are common with MELODIC: this includes the components providing authentication and authorization functionalities. It is very important these components assure the highest level of security, especially for their critical role also in the MORPHEMIC's Security system. Section 5.1 is dedicated to the components common with MELODIC.

The Web Crawler, as described in Deliverable *D3.1 Software, tools, and repositories for code mining*, is derived from MARKOS EU Project[25]: in parallel to the modifications introduced and documented in that deliverable, a security assessment has been performed in order to evaluate the needed improvements. Section 5.2 is dedicated to this topic.

The security assessment of MELODIC has been performed for integration reasons, but also to define the deployment strategy on the new testbed infrastructure (WP5). Concerning the Web Crawler, the component, detached from MARKOS context, presented some specific issues that should be analysed in detail.

Another integrated component is the ProActive Scheduler, which is a consolidated product that provides security features like user and group management using Role-based access control (RBAC) and encrypted communication using PAMR and PNPS. ProActive undergoes a continuous vulnerability assessment to maintain a high level of security. More details about ProActive's security features can be found on the specific documentation[26]. For the other components that are being developed inside the MORPHEMIC project or in other parallel project or in other parallel projects: their development process will be carried on guided by this deliverable and, more in general, according to the best security practices.

5.1 Components of the MELODIC Platform: issues and potential solutions

The analysis of security features and limitations of the MEODIC platform was performed during the first year of the MORPHEMIC project. It has been considered an important starting step to harvest the information needed to increase the security level of the inherited components for the following implementation and integration phases.

The reference version of MELODIC was the 3.0, which was the most recent one in the first year of the MORPHEMIC Project. The software architecture included a set of microservices running on Docker[27] and opening a number of ports. Specifically, the microservices presented the following features:

- thirty-two services exposing fifty-five network ports
- thirty-three network ports had to be publicly accessible for MELODIC to be functional.

Unfortunately, the aforementioned features present issues, since they extend the potential area of malicious attacks. The reduction of the numbers of open ports, the enforcement of non-public intra-component traffic going exclusively through

[25] https://cordis.europa.eu/article/id/117369-markos-global-integrated-and-searchable-opensource-software
[26] https://doc.activeeon.com/
[27] https://www.docker.com/

an internal network such as Docker's and the introduction of a proxy in order to centralise and manage the incoming web traffic are some of the objectives of the next releases of MELODIC, which will be used in MORPHEMIC.

The configuration of the services included in the platform is another critical point in terms of security. The analysis of MELODIC identified the following issues:

- Cloudiator's Web UI exposes the configured user string employed to authenticate with a Cloud Provider without any authorization needed; this string includes the configured username and Tenant ID in case of an OpenStack-based Cloud Provider
- most services are derived from stock images stored on a Docker repository, some of which come with either default users or a publicly accessible wizard to create an administrative user at first login. Moreover, the MELODIC official installation guide does not provide any indication about the services needing manual setup after installation
- private data (e.g., the credentials used to access cloud providers) is locally logged in clear text by the discovery-agent when authenticating with OpenStack-based cloud providers
- compatibility with SELinux - or similar software such as AppArmor - is not tested or documented in MELODIC; SELinux helps mitigate the risk of privilege escalation when a service gets compromised

Since Cloudiator is not present inside the MELODIC version used by MORPHEMIC and Cloudiator's role is taken by the ProActive Scheduler, the security issues concerning Cloudiator are eliminated going forward with MORPHEMIC.

Concerning the other generic considerations, a more specific analysis has been performed once the testbed has been set-up (Deliverable 5.6), and the first version of the MORPHEMIC core platform has been deployed. In particular, a vulnerability assessment on the public-facing services has been produced by Engineering for internal consumption by using OpenVAS[28]. OpenVAS is a leading free and open-source vulnerability assessment software suite aimed at offering the toolset required to evaluate the vulnerability level of networked services. It automatically scans for vulnerabilities and generates a detailed report containing the exploitable weaknesses found, while suggesting the appropriate mitigation actions. The results of this assessment show that most of the identified issues related to the platform are due to the high number of exposed services. The most significant suggestions provided by OpenVAS concern software updates or configuration tweaking - Table 6 summarizes the actions that should be taken to address the identified issues.

[28] https://www.openvas.org/

Table 6 Security issues and suggested actions found by OpenVAS on the MORPHEMIC core platform

Issue	Suggested Action
Service "insecure by default configuration"	Generate or ask for credentials during the deployment: if this is not possible at that stage, include specific post-installation actions in the installation manual
Security related post-installation procedures not executed by default	Guarantee the execution of all the security related post installation procedures
Service that does not preserve private information	Make the service private or hide it through a reverse proxy
Service logs in clear text username/passwords	Modify the logging configurations

In general, one of the best security practices is the identification of which service must absolutely be exposed to cut down the number of publicly exposed services. As mentioned above, thirty-three ports were opened in the version of MELODIC that was analysed in Y1. As such, to reduce the potential security impact, a reverse proxy has been included as part of the technological architecture of MORPHEMIC. The implementation of the reverse proxy component will be added to the MORPHEMIC platform and all the incoming web traffic will be handled by this component.

Reverse proxies can act as a single access point toward multiple hosted services. Keeping a reverse proxy up-to-date and patched and exposing only the bare minimum of ports – typically only common web traffic ports such as HTTPD (80) and HTTPS (443) - renders the proxy itself secure and, by extension, helps improving the security of the services behind it. The benefits introduced are multiple and provide a consistent uplift to MORPHEMIC security:

- having a single access point to access every exposed service, reducing the surface of attack exposed to malicious entities. While this would introduce a single point of failure in the architecture, a future investigation on how to leverage multiple reverse proxies behind one virtual IP could establish high availability at this level.
- possibility to enforce SSL encrypted network traffic on the public interfaces, increasing the security of data flowing in and out of MORPHEMIC services
- possibility to offer an additional level of protection in certain security scenarios where a reverse proxy acts as another level that needs to be penetrated before reaching the actual service backend, hence before an attacker can gain access to the underlying services.

Due to its wide adoption and free and open-source nature, the choice of software to adopt for this role has fallen on Apache Web Server[29], a very powerful and flexible web server that can be configured to act as a reverse proxy by leveraging its *mod_proxy* module. This software is extremely reliable and, as stated on the main page of the official web site, Apache Web Server is "*the most popular web server on the Internet since April 1996*". However, it should be noted that this is not a strict requirement, since other free and open-source software options that are capable of taking its place are available, such as NGINX[30].

Figure 26 summarises the proposed change in MORPHEMIC's software architecture by leveraging an Apache reverse proxy.

[29] https://httpd.apache.org/
[30] https://www.nginx.com/

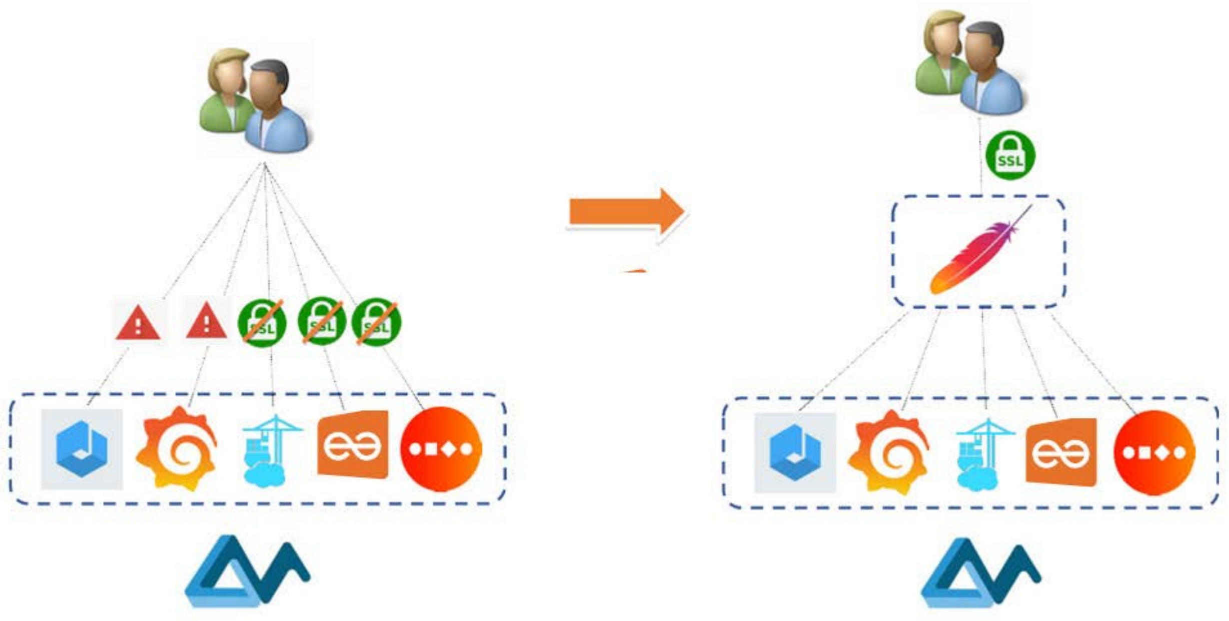

Figure 26 Adding a Reverse Proxy

5.2 Web Crawler

The first release of the MORPHEMIC Web Crawler was included in MORPHEMIC 1.0. It is based on the outcome of MARKOS, a project co-funded by EU FP7 programme. The MARKOS Web Crawler aimed at finding metadata on open-source projects stored on public repositories (such as GitHub) to allow analysis on the projects' licences. The MORPHEMIC Web Crawler inherits a specific part of MARKOS' components and adapts them to the MORPHEMIC's context and objectives. At the time of its development, within the context of MARKOS, its architecture was composed of four python modules performing the following actions:

1. remote data fetching
2. data parsing
3. data integration according to a specific logic
4. generation and storage of "Description of a Project" (or DoaP[31]) objects, which contain each identified project metadata

The second version of MORPHEMIC Web Crawler, in an advanced implementation stage at the time of writing this Deliverable, will include only the first three modules, while the latter will be part of the Knowledge Base.

Due to the fact that the Web Crawler is an in-development component not yet fully integrated with MORPHEMIC, its security has been analysed via a standalone instance. The following results will prove useful for the next releases of both the Web Crawler and the Knowledge Base - and, by extension, of MORPHEMIC – since the suggestions to address the identified issues have been already adopted and planned while such issues can match the security landscape of other MORPHEMIC components.

The Web Crawler exposes only a single public endpoint which means that it offers a small attack surface to malicious parties. Moreover, as it can be deducted from the installation process described in Deliverable 5.6, the Web Crawler runs with an unprivileged account and supports SELinux, while the basic security procedures such as, for e.g., removing guest access from MariaDB test tables, are enacted during its setup. Nonetheless, a number of security issues were identified, such as:

[31] https://github.com/ewilderj/doap

- the process (pserve) that publicly exposes the Web Crawler API does not support encryption nor authentication out of the box and it is to be used for development only
- the Web Crawler depends on Python 2.7, which has been deprecated, while it requires a few older Python libraries as well in order to execute properly
- some sensitive data, such as the username/password combination required to access the crawler database, is stored in clear text inside the crawler configuration file.

In order to improve the security of this software, the following changes have been implemented:

- the adoption of ElasticSearch, and the use of its API to query the data mined from the Web Crawler, has enabled:
  - the removal of the insecure pserve web server
  - the reuse of its A.A.A. facilities, configured to work in conjunction with the Login Client, thus offering an authentication mechanism to limit the access to the Web Crawler/Knowledge Base API

While the following enhancements are planned to go forward:

- API endpoint access to be restricted to the Docker internal network, in order to limit it to intra-process communication
- port the Web Crawler to Python 3.9 to increase its security and software support as well as replacing all legacy libraries that are or will be unsupported going forward
- In order to protect sensitive configuration data such as username and password required by the Web Crawler to access its database, implement a mechanism that either lets the crawler ask for the database password at start-up or hash/encrypt the file containing the required authentication data and let the crawler decrypt it at boot. This is comparable to how X11vnc[32] deals with session passwords. The latter solution would make for a more practical choice as it does not require user intervention. This would represent an improvement upon clear-text username/password storage while maintaining the benefits of running the process under an unprivileged user: in fact, if a service that runs as root is exploited, the whole system is compromised. Meanwhile, if the crawler is exploited, the malicious user could gain access to its files and database, but he would still need to find a way to elevate his privileges.

At time of writing this deliverable, the development of the Web Crawler is actively ongoing. The technical team of the project is replacing the original prototype of the Knowledge Base, derived by MARKOS Crawler, with an ElasticSearch based solution, while also increasing the exhibited security level by adopting the measures explained above. In fact, the aforementioned suggestions have been included in the development plan of the Web Crawler and will be implemented in the next releases.

[32] https://github.com/LibVNC/x11vnc

6 Conclusions

Security in MORPHEMIC is intended in two different meanings: Platform Security (Chapter 3) and Deployment Security (Chapter 4).

The Platform Security includes all the functionalities, services and mechanisms aimed at making the platform *secure*. The Authentication and Authorization are the basic security services: they are derived from the corresponding services of MELODIC. Concerning the Authentication, MORPHEMIC acts both as server and as client. MORPHEMIC is a server for the User-Authentication (Section 3.1.1): it is based on username and password verified against identities stored on an LDAP. A successful authentication produces a signed *token* that identifies each request inside the platform. MORPHEMIC is a client with respect to the Authentication mechanisms of the Deployment Environments. In particular, MORPHEMIC must log into the Deployment Environment that it has to manage: the ProActive Scheduler handles this aspect supporting different authentication factors and providing a credential storage service.

The Authorization system is derived by MELODIC as well: however, even if it is similar concerning the operations and the used protocols (ABAC and RBAC), in MORPHEMIC the technological architecture is simplified, since the whole system is built on Balana XACML implementation. The Authorization system is classified in RBAC User-Authorization, aimed at controlling the access to the services provided by the platform, and ABAC Deployment Environment Pre-Authorization, aimed at controlling the accesses to the Deployment Environments.

The Access Logger has been introduced in MORPHEMIC: it records all the events relevant in terms of security. It is an agent-based component that gets the records from the Authentication and the Authorization system and stores them.

The Platform Security aims at reducing the risk of potential attacks: however, another important issue is due to the specific vulnerabilities of the components (Chapter 5). An analysis on the components common with MELODIC and on MORPHEMIC's Web Crawler has been performed and the results have been used to plan some platform/component improvements to be introduced in the next MORPHEMIC platform releases.

The Deployment Security assures that the selected Deployment Environments and nodes present a security level adequate to the requirements of the specific application. In MORPHEMIC this functionality is implemented by introducing the security requirements in the CAMEL Model, Utility Function and Reasoning (Section 4.4) and filtering the Deployment Environments and nodes (Section 4.3) according to a set of pre-defined security parameters (Section 4.2). The *node selection problem* will consider also the requested security level and the final result will optimize the Utility Function taking into account also this aspect.

This deliverable provided a complete description of the security definitions adopted in MORPHEMIC and the architecture to implement it. At the time of writing the document the implementation is still in progress: security is a complex topic and should always be *tuned* according to the provided services and new potential risks. To this end, the implementation will embrace the whole project period as component owners need to work on increasing the security level of their components according to the guidelines given in this document. The future works in terms of security will be mainly linked to the impacted components: their specific deliverables, especially the ones of WPs 1, 2, and 3 will report the future works. The Deployment Security is the answer to some requirements raised by the use-cases: the evaluation of how MORPHEMIC will meet these requirements is part of the work of WP 6 and will be documented by Deliverable 6.5 (*Validation Results*).

Furthermore, the Deployment Security is one of the most interesting research results that came out from the first period of the project. It was not planned at proposal time and has been defined thanks to the analysis of the requirements provided by the use-cases: for this reason it is not covered by the current version of the Description of Actions. However, in order to provide a guideline for the future extensions of MORPHEMIC, some tasks for the implementation of the Deployment Security have been identified:

- implementation of the Deployment Environment security profiles
- extension of the Utility mechanism
- extension of the CAMEL Model in order to include manually defined security features: this is an optional step to support some specific cases in which the *implicit* association of the security features (Section 2.3) is not sufficient.

References

[1] Paweł Skrzypek, Yiannis, Verginadis, Ioannis Patiniotakis, and Christos Chalaris, ‘Melodic Deliverable 5.03 Security requirements & design’. [Online]. Available: https://melodic.cloud/wp-content/uploads/2020/03/D5.03-Security-requirements-design.pdf

[2] M. Jones, J.Bradley, N.Sakimura, ‘RFC 7519’. Internet Engineering Task Force (IETF), 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

[3] Y.Sheffer, D.Hardt, M.Jones, ‘RFC 8725’. Internet Engineering Task Force (IETF), 2020. [Online]. Available: https://tools.ietf.org/html/rfc8725

[4] Chung Tong Hu, David F. Ferraiolo, David R. Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, ‘Guide to Attribute Based Access Control (ABAC) Definition and Considerations’. NIST Special Publication, Feb. 25, 2019. [Online]. Available: https://www.nist.gov/publications/guide-attribute-based-access-control-abac-definition-and-considerations-1

[5] Chad R. Dougherty, K. S. Kirk, Robert Seacord, David Svoboda, and Kazuya Togashi, ‘Secure Design Patterns’, 2018, doi: https://doi.org/10.1184/R1/6583640.v1.

[6] Dan Bergh Johnsson and Daniel Sawano, *Secure by Design*. Manning.