Model and Inference Driven Automated testing of Services architectures

Deliverable D6.2

Specification and design of the basic MIDAS platform as a service on the Cloud
## Revision Chart

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author (Partner)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>28/11/2013</td>
<td>CNR</td>
<td>Table of Content Definition</td>
</tr>
<tr>
<td>0.2</td>
<td>31/12/2013</td>
<td>CNR, SEF</td>
<td>Preliminary version of sections 1, 2, 3</td>
</tr>
<tr>
<td>0.3</td>
<td>30/01/2014</td>
<td>CNR</td>
<td>Preliminary version of section 4 and 5</td>
</tr>
<tr>
<td>0.4</td>
<td>31/01/2014</td>
<td>CNR</td>
<td>Preliminary draft for TMC discussions</td>
</tr>
<tr>
<td>0.5</td>
<td>10/02/2014</td>
<td>CNR, SEF, FF</td>
<td>TMC discussion integration</td>
</tr>
<tr>
<td>0.6</td>
<td>12/02/2014</td>
<td>CNR</td>
<td>Cost analysis added, fixed introduction</td>
</tr>
<tr>
<td>0.7</td>
<td>14/02/2014</td>
<td>CNR, SEF</td>
<td>Appendix A added</td>
</tr>
<tr>
<td>0.8</td>
<td>14/02/2014</td>
<td>CNR</td>
<td>Minor corrections and fixed references.</td>
</tr>
<tr>
<td>0.9</td>
<td>14/02/2014</td>
<td>CNR</td>
<td>Version distributed for internal review</td>
</tr>
<tr>
<td>1.0</td>
<td>25/02/2014</td>
<td>CNR</td>
<td>Final version revised according to the comments received by internal reviewers</td>
</tr>
</tbody>
</table>
**EXECUTIVE SUMMARY**

The MIDAS Testing as a Service (MIDAS TaaS) platform is designed and architected according to a SOA-based approach, and provided as an integrated Testing as a Service framework available on demand, i.e., on a self-provisioning, pay-per-use, elastic basis.

For this reason the MIDAS platform is deployed on the Cloud, in particular on the public Cloud IaaS platform Amazon EC2, as discussed in D6.1 [1]. The platform is made accessible as a multi-tenancy SaaS (Software as a Service) from an end user perspective. The costs saving and easy accessibility of Cloud's extremely large computing resources will make the MIDAS testing facility usage available to geographically distributed users, executing wide varieties of user scenarios, with a scalability range previously unattainable in traditional testing environments.

In order to design the strategy to deploy the MIDAS platform on the Cloud, the classes of MIDAS potential users, their roles, as described in D2.2 [3], and their interactions with the MIDAS platform are further analysed and detailed by describing the main use cases for the identified classes. Then, the main MIDAS services needed to cover all the analysed use cases are reported. This analysis allowed us to identify the building blocks of the basic MIDAS platform and their interactions that are described in terms of sequence diagrams in the document. Furthermore, a specification of the APIs of ancillary MIDAS services for MIDAS administration and MIDAS storage management are defined.

This step allowed us to have a clear vision of the Cloud resources and the Cloud services, necessary for the MIDAS platform deployment on the Cloud.

So, the design of the approach that will be adopted for its deployment on a Cloud IaaS is proposed. As discussed in D6.1 [1], IaaS Clouds allow users to have full control of the entire software environment in which their applications run. The benefits of this approach include support for legacy applications, and the ability to customize the environment to suit the application. This is a very crucial requirement for the MIDAS platform development, since it relies on components that are developed and implemented by the MIDAS technical partners independently and that have to be integrated in a complete application, so requiring software development environment that fulfil different requirements and needs in terms of software support and customization. Furthermore, MIDAS includes legacy commercial software, such as the TTWorkbench execution engine, that may require specific software support to be included in the MIDAS platform. The drawbacks of IaaS Clouds include increase complexity and additional effort required to setup and deploy the application.

We adopted a virtualization approach already in the development phase of MIDAS platform. In particular, we used VMware as the virtualization infrastructure that introduces a layer of abstraction known as a hypervisor. A hypervisor runs on top of physical hardware, allocating resources to isolated execution environments known as Virtual Machines, which run their own individual virtualised operating system. The separation between resource provision and operating systems introduced by
virtualisation technologies is a key enabler for Cloud computing, specifically for Infrastructure-as-a-Service (IaaS) Clouds.

For this reason, in order to facilitate and support the integration and testing of tools independently developed by MIDAS partners and their incremental integration on the MIDAS platform on the Cloud, a guide has been released to all MIDAS technical partners providing a detailed description of the MIDAS development environment designed by CNR to enable developers to implement, debug, and test their software components on a seamless and loosely coupled platform. The guide “Virtual Machine Image Guide: How to develop MIDAS Components v1.2” [4] is a technical document not included in the present one.

The preliminary design of the Cloud-based MIDAS platform architecture is proposed in this document, detailing the strategies adopted for its deployment on the selected Cloud infrastructure together with the description of the Cloud services of the underlying Cloud infrastructure that will be used to provide an operational platform with all the basic capabilities.

Also, the economic implications in using a public Cloud computing facility and services is evaluated providing a preliminary analysis of the costs the MIDAS consortium will incur during the remaining project life to prevent exceeding the planned economic resources.

In order to demonstrate the main functionalities of the MIDAS basic platform, we provide a preliminary experience in deploying the MIDAS execution component on the underlying Cloud infrastructure.
**ABSTRACT**

<table>
<thead>
<tr>
<th>Project Number</th>
<th>318786</th>
<th>Project</th>
<th>MIDAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>Model and Inference Driven Automated testing of Services architectures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project website</td>
<td><a href="http://www.midas-project.eu">http://www.midas-project.eu</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document URL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>D6.2</td>
<td>Specification and design of the basic MIDAS platform as a service on the Cloud</td>
</tr>
<tr>
<td>Work Package</td>
<td>Number</td>
<td>Title</td>
</tr>
<tr>
<td>Number</td>
<td>6</td>
<td>Cloud support for SOA Testing PaaS &amp; SOA testing framework integration</td>
</tr>
<tr>
<td>Task</td>
<td>Number</td>
<td>Title</td>
</tr>
<tr>
<td>Number</td>
<td>6.1</td>
<td>Automated test execution platform</td>
</tr>
<tr>
<td></td>
<td>6.3</td>
<td>Automated test execution environment</td>
</tr>
<tr>
<td></td>
<td>6.5</td>
<td>SOA testing framework integration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of Delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>draft</td>
<td>final</td>
</tr>
<tr>
<td>Dissemination</td>
<td>Public</td>
<td>Confidential</td>
</tr>
<tr>
<td>Level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary for</td>
<td>This deliverable reports the deployment strategy of the basic MIDAS TaaS on the Amazon EC2 public Cloud infrastructure</td>
<td></td>
</tr>
<tr>
<td>dissemination</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>Cloud computing, Elasticity, Virtual Machine, SOA Testing, Web Services, Test Execution</td>
<td></td>
</tr>
<tr>
<td>Contact Person</td>
<td>Nicola Tonellotto (CNR)</td>
<td></td>
</tr>
</tbody>
</table>

**Authors (Partner)**
Nicola Tonellotto, Claudia Di Napoli, Alberto De Francesco, Maurizio Giordano (CNR), Fabio De Rosa (SEF), Fabio De Rosa (SEF), Libero Maesano (SEF)

**Reviewers**
FF, T6
# Table of Contents

**Executive summary** ........................................................................................................ 3

**Abstract** ................................................................................................................................ 5

**Table of contents** .................................................................................................................. 6

**1 Introduction** .......................................................................................................................... 8
  1.1 Project Overview .................................................................................................................. 8
  1.2 Purpose of the document ..................................................................................................... 8
  1.3 Document structure ............................................................................................................. 9

**2 MIDAS Testing As A Service Platform** ............................................................................... 10
  2.1 MIDAS Users .................................................................................................................... 10
  2.2 Use Cases .......................................................................................................................... 12
    2.2.1 End user use cases ....................................................................................................... 12
    2.2.2 Tenancy admin use cases ............................................................................................ 14
    2.2.3 TaaS admin use cases ................................................................................................. 15
  2.3 End User and Core Services ............................................................................................... 16
  2.4 Tenancy Admin Services .................................................................................................... 18
  2.5 The basic MIDAS platform ............................................................................................... 19

**3 Use Case Sequence Diagrams** ........................................................................................... 21
  3.1 Tenancy Admin Use Cases Sequence Diagrams ................................................................ 21
    3.1.1 I&A Sequence Diagrams ............................................................................................. 21
    3.1.2 A&B Sequence Diagram ............................................................................................. 22
  3.2 End User Main Use Cases Diagrams .................................................................................. 23
    3.3.1 Direct Test Execution Sequence Diagram ..................................................................... 24
    3.3.2 Manual Test Design Sequence Diagram ..................................................................... 27
    3.3.3 Automated Test Design Sequence Diagram .................................................................. 29

**4 Tenancy Admin APIs** .......................................................................................................... 31
  4.1 Identification & Authentication Service .............................................................................. 31
    4.1.1 Create User .................................................................................................................. 32
    4.1.2 Delete User .................................................................................................................. 33
    4.1.3 List Users .................................................................................................................... 34
    4.1.4 Verify User .................................................................................................................. 34
  4.2 Accounting & Billing Service .............................................................................................. 35
    4.2.1 Get Billing Information ............................................................................................... 36

**5 MIDAS Platform on the Cloud** ........................................................................................... 38
  5.1 Amazon Cloud solutions for MIDAS .................................................................................. 38
    5.1.1 Cloud/Computing Building blocks ............................................................................. 38
    5.1.2 Elastic Computing Service ........................................................................................... 39
    5.1.3 Storage Services ......................................................................................................... 39
  5.2 MIDAS Deployment on Amazon Cloud .............................................................................. 40
    5.2.1 MIDAS Building Blocks ............................................................................................ 42
    5.2.2 MIDAS Computing Elasticity ..................................................................................... 43
    5.2.3 MIDAS Storage Scalability and Redundancy ............................................................ 44
    5.2.4 Preliminary Cloud Cost Analysis ............................................................................... 45

**6 Conclusions** ....................................................................................................................... 47
A Basic Architecture For the Integrated MIDAS Testing Platform on the Cloud. ...... 48
References ........................................................................................................................................... 52
1 INTRODUCTION

1.1 PROJECT OVERVIEW

The MIDAS project aims at designing and building an integrated framework for Service Oriented Architecture (SOA) testing automation that will be available as a Software as a Service (SaaS) on a Cloud infrastructure and that spans all the testing activities: test generation, execution, evaluation and scheduling, on the functional, interaction, fault tolerance, security and usage-based testing aspects. MIDAS is focused on SOA testing, i.e. on black-box testing of single services and on grey-box testing of services architectures. The testing methods and technologies that are investigated and prototyped in the project are beyond the state of the art, particularly on model-based testing, model checking of choreographies for sound interaction test scenarios, fuzzing for security testing, usage-based testing, probabilistic inference reasoning about test evaluation and scheduling.

In order to evaluate the effectiveness and the usability of the MIDAS framework facilities, two pilots will be used for SOA testing in different real business domains: Healthcare (HC) and Supply Chain Management (SCM).

1.2 PURPOSE OF THE DOCUMENT

This document is the second deliverable released within the Work Package 6 “Cloud support for SOA Testing PaaS & SOA testing framework integration”. Its purpose is to provide a description of the specification and design of the MIDAS platform components that are being developed for the deployment of the basic MIDAS platform as a service (MIDAS TaaS) on a public Cloud infrastructure.

The deliverable D6.1 provided the technological basis to select first a suitable Cloud model, and then a suitable public Cloud infrastructure for the MIDAS deployment on the Cloud.

In the present document the classes of MIDAS users and the corresponding use cases identified so far are introduced. Then, the additional use cases necessary to provide the complete set of functionalities for the deployment of the basic MIDAS platform on the Cloud are detailed together with the corresponding MIDAS services necessary to provide the identified functionalities.

So, the building blocks of the MIDAS platform and their interactions are identified in accordance with the MIDAS requirements reported in D2.1 [2] and the MIDAS architecture reported in D2.2. Also, the APIs for the additional ancillary services, not included in D2.2, are defined.

Then, the initial design of the Cloud-based MIDAS platform architecture is proposed detailing the strategies adopted for its deployment on the selected Cloud infrastructure together with the description of the Cloud services of the underlying Cloud infrastructure that will be used to provide an operational platform with all the basic capabilities.

Finally, a preliminary cost analysis of the Cloud resources the MIDAS consortium will incur during the remaining project life is reported to prevent potential economic criticality in advance.
In addition, we provide a preliminary experience in deploying the MIDAS execution component on the underlying Cloud infrastructure to demonstrate the main functionality of the basic MIDAS platform on the Cloud.

According to the DoW, this is a public deliverable, so it has a twofold intention: on one hand it provides the MIDAS partners (both technical and user ones) with an insight of the deployment strategy adopted for the MIDAS platform on a public IaaS Cloud, together with the necessary Cloud services to meet its functional and software requirements, and an overview of the foreseen interactions of the different users with the MIDAS platform; on the other hand it may target also Cloud developers outside the MIDAS Consortium by describing the adopted methodology to deploy SOA-based software testing applications on the Cloud, together with preliminary economic considerations on the incurred costs.

1.3 DOCUMENT STRUCTURE

The rest of the document, following Section 1 reports an overview of the MIDAS project, the purpose of the present document, and its structure, is organized in 5 sections:

• Section 2 presents an overview of the classes of users for the MIDAS TaaS and the corresponding use cases. Then, the main MIDAS services, as identified in D2.2, are recalled and extended with additional use cases. Then a description of the MIDAS services that will be included in the basic MIDAS platform on the Cloud are reported with some considerations on the ones that will be not part of the current implementation.

• Section 3 provides a description of the interactions occurring for the use cases identified so far that will be part of the first MIDAS prototype, in the form of sequence diagrams.

• Section 4 reports the specifications of the APIs for the additional ancillary services that were defined in the previous section.

• Section 5 provides an overview of the main Cloud services of the underlying public Cloud infrastructure that will be used by the MIDAS platform, followed by a description of the strategy adopted for the deployment of MIDAS on the Amazon Cloud infrastructure, discussing its rationale. Finally, a preliminary evaluation of the cost concerning the use of Cloud resources, both in terms of computing facilities and services, is reported.

• Appendix A reports the preliminary experience in deploying the MIDAS execution component on the underlying Cloud infrastructure.
2 MIDAS TESTING AS A SERVICE PLATFORM

In this section, we briefly report an overview of the MIDAS users according to their roles and tasks, together with the use cases for each class of identified users. Furthermore, the MIDAS platform specifications, according to the architecture reported in the deliverable D2.2, and a high level description of MIDAS user and core services, that constitute the building blocks of the basic MIDAS platform deployed as a service on the Cloud, are reported. Moreover, we analyse and extend the use cases of end users, tenancy administrators and TaaS administrator to specify and discuss ancillary and non-functional capabilities of the MIDAS platform.

2.1 MIDAS USERS

As reported in D2.2, the MIDAS Platform, referred to as MIDAS TaaS, has four classes of users, each one playing different roles in interacting with the platform:

1. End users: they consist of both users responsible for planning, designing, and conducting test campaigns on service architectures (also known as testers), and users responsible for the creation, administration, and deployment of the service architecture under test (also known as testees).
2. Test method developers: they consist of users responsible for designing and implementing test methods to be used for conducting test campaigns.
3. Administrators: they consist of users responsible for managing both the identification and authentication of end users and test method developers, and the MIDAS TaaS facilities used by the administered users, including the accounting and billing of these facilities.
4. TaaS administrator: it is the responsible entity for the entire TaaS platform, and for managing any interaction with the selected Cloud provider of the IAAS platform for the MIDAS TaaS development and operation. As such, he is responsible for the dynamic provisioning of all MIDAS public functionalities and for configuring the underlying Cloud resources and services for MIDAS TaaS users, and for any interaction with the Cloud IAAS provider.

End users and test method developers are conceptually grouped in logical facilities called respectively tenancies and labs that are users computing spaces managed by tenancy/lab administrators. Tenancies and labs are unit of:

1. end users identification and authentication;
2. Cloud resources allocation, accounting and billing;
3. data and services ownership and access.

According to the MIDAS requirements, each tenancy/lab must be able to deal with its own Cloud users, Cloud resources, data and services in a private, sandboxed way.

The composition relationships and interactions among users and facilities of the MIDAS platform are reported in Figure 2.1. As shown, the MIDAS platform may contain several tenancies (resp. labs), each one composed of several end users (resp. test method developers). Each tenancy (resp. lab) is managed by a tenancy admin (resp.
lab admin) that interacts with the respective users, and it is responsible for creating user accounts and credentials for them.

According to the requirements of the MIDAS platform and its design, it is assumed that the underlying Cloud infrastructure/middleware is completely transparent to MIDAS end users, while tenancy/lab admins are aware only of Cloud resources usage and billing, but they are not involved in their management/allocation.

![Diagram of MIDAS user interactions](image)

**Figure 2.1 - Composition relationships and interactions among MIDAS users**

The MIDAS platform is provided and managed by a single entity, the TaaS admin, also known as the MIDAS provider. It is the only one responsible for:

- creating, deploying, managing, and disposing tenancies/labs on the MIDAS platform
- interacting with the provider of the underlying Cloud infrastructure,
- establishing and enforcing the rules of configuration of Cloud resources and services for each tenancy/lab,
- monitoring the deployment of the MIDAS services (end user, core and admin services) for each tenancy/lab.

So all the Cloud infrastructure features are completely hidden behind the MIDAS provider. All MIDAS users just interact with the respective public APIs and services, published on the Internet through the MIDAS platform by the MIDAS TaaS admin. The TaaS admin, in general, fixes the rules to use the MIDAS platform underlying resources and monitor their usage by the user applications.

It is assumed that before the creation of a tenancy/lab, the TaaS admin interacts with the tenancy/lab admins to establish service level agreements (SLAs), i.e. legal contracts, between himself, as the MIDAS service provider, and the tenancy/lab admins as MIDAS services consumers, where regulations, duties, payment policies concerning the usage of MIDAS services and computing resources are stated. The MIDAS TaaS offers one or a small number of “standard contracts”. Hence, we envision a contract template as a pricing mechanism (how the tenancy/lab pays for the MIDAS TaaS services) coupled with a resource allocation policy (how the MIDAS TaaS admin
pays the Cloud provider). For the time being, the specification of the pricing mechanism is beyond the scope of the project, and so outside the scope of this deliverable. Conversely, the resource allocation policy depends on the Cloud provider. Since the selected Cloud provider for the project is Amazon Web Services, this policy, for the duration of the project, is fixed, and depends on the Amazon services the MIDAS TaaS will use, as discussed in Sec. 5.

In the rest of the document, we will concentrate our focus on the end users and the tenancies for what concerns the deployment on the Cloud. In fact, the first MIDAS TaaS prototype will not deal with labs and test method developers, according to the agreed incremental and stepwise development of the MIDAS platform. From the Cloud deployment point of view, the MIDAS TaaS platform on the Cloud can be easily extended with labs, as discussed in Section 2.5.

2.2 Use Cases

The MIDAS TaaS functionalities are described through use cases, organized around the different typologies of MIDAS users. For the time being, the use cases described in the following subsections refer to the MIDAS end user, and to the MIDAS tenancy admins since test method developers and lab admins are not included in the MIDAS basic platform.

In addition, the use cases of the TaaS admin are reported even though the functionalities to support their implementation will not be included in the MIDAS TaaS platform, since for the time being the TaaS admin, as MIDAS provider, will execute its tasks manually and outside the MIDAS TaaS platform.

Finally, use cases crossing different types of users, are reported as they are necessary to support all the described MIDAS use cases.

2.2.1 End user use cases

The end user (main) use cases as detailed in the deliverable D2.2 are:

1. Direct Test Execution, consisting in the execution of MIDAS-compliant legacy TTCN-3 test suites;
2. Manual Test Design, consisting in the execution of test cases and data provided in a non-executable and platform independent model;
3. Automated Test Design, consisting in the automatic generation of test cases and data, and their execution.

They are sketched briefly in Figure 2.2 with the additional use case Identity & Authentication used to check that each end user is a registered user of that tenancy, and it is authenticated before invoking the capabilities of that tenancy. The authentication, in general, will be propagated to the whole MIDAS architecture to identify and authenticate the end users with the other tenancy services, as well as with MIDAS core services. As this aspect represents a cross-cutting concern among all MIDAS services, it is included and used automatically in all end user use cases.
In Figure 2.3 we introduce and detail new end user use cases, the **end user (ancillary) use cases**, directly related with the file management activities for the persistent storage space for storing user data within a tenancy, and test method inspection activities to list test methods that are available at the level of the entire MIDAS TaaS platform. Also these uses cases rely again on the Identity & Authentication use case to correctly authenticate the end users.
2.2.2 Tenancy admin use cases

The tenancy admin use cases are reported in Figure 2.4. The tenancy admin role extends the end user role, i.e., he has access to the end user functionalities, but has access also to a collection of administrative functionalities:

- end user identification and authentication: tenancy admins are in charge of creating and deleting their tenancy end users;
- accounting and billing: tenancy admin are in charge of paying for the resource usage among all end users of their tenancy, and are able to access information about the current resource consumption and the MIDAS platform billing costs at any time.

The tenancy admin is enabled by the TaaS admin when the tenancy is first created, and it is disabled by the TaaS admin when the tenancy is dismissed. As already mentioned, the interactions between a tenancy admin and the TaaS admin must be regulated by a specific agreement, established before the tenancy is first created. The tenancy admin is also the first end user of the tenancy, but must log in with self-created end user credentials to access the end user capabilities.
Lab admins have the same use cases and similar interactions with the TaaS admin, with the only difference that they are the first test method developer of the respective lab, and as such, they extends the test method developer user role.

### 2.2.3 TaaS admin use cases

The TaaS admin use cases are reported in Figure 2.5, where its interactions with the tenancy/lab admins and the Cloud infrastructure provider to perform his tasks are shown. The TaaS admin use cases include:

1. the tenancy creation, management and shutdown according to the interactions of the TaaS admin with the tenancy admin; these duties are performed through the deployment and provisioning mechanisms offered by the Cloud provider, as well as additional TaaS administration services that might be necessary; moreover, the TaaS admin manages the identity and authentication of the tenancy admin;

2. the management of the test methods in the **MIDAS portfolio**, a shared repository of test methods available to all tenancies in the MIDAS platform; this management includes the addition of new test methods to the MIDAS portfolio (usually developed by test method developers of a lab), the removal of existing test methods from the MIDAS portfolio and the listing of all available test methods;

3. the accounting of Cloud resources used by all tenancies and labs of the MIDAS platform, in order to track the costs of testing activities performed by all users of a specific tenancy.

4. The billing of the tenancy on the basis of the MIDAS pricing model.
2.3 END USER AND CORE SERVICES

The main functionalities of end users are offered by three end user services, depicted in Figure 2.6, that are able to support all the end user use cases. They are:

1. Test Gen&Run Service, it allows to asynchronously start the execution of a test task (either a test generation or a test execution task), and to actively poll it to inspect the status and the resulting output of any started test task;

2. Test Method Query Service, it allows end users to list the test methods currently part of the MIDAS portfolio, and to retrieve the properties of any method in the MIDAS portfolio; all its methods are synchronous;

3. File Management Service, it offers access to the file system private to the tenancy the end user belongs to, and to perform the usual operations supported by a file system.

The MIDAS architecture presented in D2.2 provides the APIs for the implementation of the Test Method Query Service and the File Management Service, and their implementation will be described later on in this deliverable since it is directly mapped on the Cloud infrastructure relying on facilities provided by the Cloud provider.

A more detailed description of the implementation of the Test Gen&Run Service was given in D2.2 where its design was specified. The Test Gen&Run Service is composed of several services, also referred to as the MIDAS core services, that...
contribute to the implementation of the end user test generation and execution functionalities. The core services are not exposed by the MIDAS platform, but they allow test method developers to implement specific test methods. The Test Gen&Run Service implementations will allow end users to use these test methods.

The core services of MIDAS are organised in two levels, as depicted in Figure 2.6. The first level distinguishes the Test Generation Service from the Test Run Service. While the first service is responsible for automatically generating test cases, test scripts and model transformations for testing, the second service coordinates the run of a specific test cycle, organized in three phases, as described in D2.2: an optional scheduling phase, a mandatory execution phase, and an optional arbitration phase.

According to the specification reported in the D2.2, the Test Generation Service (resp. the Test Run Service) is provided by a Test Generation Container (resp. a Test Run Container). Each container can include different modules as plug-ins, each of them implementing a specific test generation (resp. test run) capability, with the same interface of the test generation service (resp. test run service). Both Test Gen and Run Services are invoked asynchronously, and their outcome is notified to the Test Gen&Run Service through a notification, whose listener is provided by the Test Gen&Run Service.

The second level of the Test Gen&Run Service architecture concerns the Test Run Service. It includes three independent services: the Test Arbitration Service, the Test Scheduling Service, and the Test Executor Service. These services are provided by a corresponding container, as for the Test Generation Service and the Run Service. Also for these services, each container can include different modules as plug-ins, each of them implementing a specific capability, with the same interface of the corresponding service. All services in the second level expose just two methods, one to initialise the corresponding service, and one to actually execute the provided service. Both methods of the three services are invoked asynchronously, and their outcome is notified to the Test Run Service through a notification, whose listener is provided by the Test Run Service.

Figure 2.6 - End User and Core Services
2.4 Tenancy Admin Services

The main functionalities of tenancy admins are offered by two tenancy admin services, as depicted in Figure 2.7:

1. Identification & Authentication Service (I&A Service), it allows tenancy admins to create and delete tenancy end users, as well as to list current members of a tenancy, and to verify that each member of a tenancy is authenticated before invoking the facilities of that tenancy;

2. Accounting & Billing Service (A&B Service), it allows tenancy admins to monitor the resource usage of the MIDAS facilities and services of the tenancy as a whole, and to get updated and consolidated billing information about the MIDAS services at periodic intervals.

The I&A Service is one of the cross-cutting services of the MIDAS TaaS platform since it is used by all MIDAS services requiring authentication of end users before they perform any operation. For example, the Test Gen&Run Service should be invoked by an end user providing authentication information (e.g., username and password) that must be checked by the I&A Service before the invoked service can perform the requested operation.

As already mentioned, tenancy admins, as all MIDAS users, cannot directly access any Cloud-related service of the underlying Cloud infrastructure used for the deployment of the MIDAS TaaS platform. So, the A&B Service is used only to inform the tenancy admin on how the end users of the tenancy are exploiting the MIDAS facilities in terms of services and computing resources according to what agreed with the TaaS admin before the tenancy is created. Finally, the end user and the tenancy admin services detailed so far will be exposed on the Internet in a programmatic way through well defined APIs by the MIDAS gateway (see Figure 2.7), responsible for hosting the services implementation and for providing mechanisms to invoke them using HTTP. Since all services must be secured independently from their functionalities and implementation, we leverage HTTP basic security to enforce access control to the MIDAS services. This means that the MIDAS gateway will be also responsible for enforcing access controls to MIDAS services based on static, standard HTTP headers (which means that no handshakes have to be done in anticipation). The “basic” authentication scheme is based on the model that the client must authenticate itself with a username and a password for each service invocation. The basic authentication scheme provides no confidentiality protection for the transmitted credentials. If necessary, basic authentication could be used over encrypted connections such as HTTPS, leveraging TLS/SSL protocols.
2.5 **THE BASIC MIDAS PLATFORM**

We reported an overview of main use cases for the different users of the MIDAS TaaS platform, together with the ancillary use cases necessary to support all MIDAS functionalities as they are described in D2.2.

We recall here that the services corresponding to the reported use cases were described only for end users and tenancy administrators since the first MIDAS TaaS prototype will not deal with labs and test method developers, according to the agreed incremental and stepwise development of the MIDAS platform reported in the Integration Plan [5].

Issues/problems/criticality concerning the implementation and deployment of the MIDAS platform on the Cloud have been analysed both for end users/tenancies and test method developers/labs, and they result equivalent from the Cloud point of view. So, once the first prototype will be released, it will be easy to adapt the architecture, deployment and management of a tenancy and the respective end users on the Cloud, also to labs and the respective test method developers.

It should be noted that the accounting procedures of the actual Cloud resources and services depend on the service level agreement that will be established between the TaaS admin (the MIDAS services provider) and each tenancy admin (the MIDAS service consumers representative). For example, while the TaaS admin could be in charge of paying all the Cloud resources of a deployed MIDAS platform to a selected Cloud provider, he/she can charge these costs to the tenancy admins as operational costs of the tenancy, plus variable costs representing its margin of profit.

---

*Figure 2.7 - MIDAS public services, APIs and gateway*
These aspects will be left aside, since the nature and the contents of the agreements between the TaaS admin and the tenancy/lab admins can be dealt with at the MIDAS industrialization phase only when the economic and strategic evaluation of the potential use of the MIDAS TaaS facilities on the market will be addressed. So at the moment, basic information on Cloud resources usage will be made available to tenancy/lab admins.
3 USE CASE SEQUENCE DIAGRAMS

In this section we design and illustrate the sequence diagrams for the end user and tenancy admin use cases reported in Section 1, showing the interactions among instances of end user, core and tenancy admin services. The interactions for the reported use cases have been specified in D2.2, and they are here reported in the form of sequence diagrams.

The sequence diagrams for MIDAS tenancy admin use cases are detailed first, in order to provide a complete view on the interactions among the MIDAS tenancy building blocks and the sequence of messages exchanged between components needed to carry out the core functionalities of any MIDAS tenancy. Then the end user ancillary use cases are briefly discussed; they are necessary for any MIDAS end user main use case, as their correct behaviour is mandatory for any of them. Eventually, we detail and discuss the end user main use cases. The detailed knowledge of the interactions among components and their time ordering is of paramount importance to design the MIDAS services implementations and to correctly deploy them on any runtime execution environment, in particular on a Cloud.

3.1 TENANCY ADMIN USE CASES SEQUENCE DIAGRAMS

The tenancy admin use cases involve interactions between tenancy admin services (i.e., I&A and A&B services) and the tenancy administrator. One additional sequence diagram is reported, not related to any tenancy admin use case of Figure 2.4, reporting an internal interaction of the I&A service with the MIDAS components requiring the verification of the credentials supplied by a tenancy end user to access end user/core services when they are invoked.

3.1.1 I&A Sequence Diagrams

The I&A sequence diagrams involving direct interaction with the tenancy admin (mediated, if necessary by the MIDAS gateway/portal) are reported in Figure 3.1. They concern respectively the creation/deletion of a tenancy end user, and the listing of all tenancy end users.

![Sequence Diagrams](image.png)

*Figure 3.1: I&A sequence diagrams involving a tenancy admin*
The interactions are all synchronous, message-passing operations between the tenancy admin and the I&A service, whose operations are backed up in some implementation-dependent database responsible to provide reliable storage options to the implementation of the I&A Service.

The last I&A sequence diagram, reported in Figure 3.2, depicts the interaction of any MIDAS service (end user or core service) with the I&A service to verify the credentials of the end user invoking the service. In this scenario, the invoked service has received, through an authenticated invocation, the credentials of a user, and the service needs to verify them (i.e., check the username exists and the corresponding password is correct) before the actual execution of the invoked functionality.

![Figure 3.2: I&A sequence diagram involving MIDAS services](image)

Note that this verification happens for all end user services that must always be accessed in a secure way. The core services invocations will not be authenticated explicitly, as they are accessed through an authorised end user service only. However, we assume that also core services “might” require access to this verification procedure from time to time, e.g., to backlog authenticated user activities, or to be directly invoked by an authenticated test method developer in a specific lab.

### 3.1.2 A&B Sequence Diagram

The A&B only sequence diagram involves a direct interaction with the tenancy admin (mediated, if necessary, by the MIDAS gateway/portal), and it is reported in Figure 3.3.
Figure 3.3: A&B sequence diagram

The A&B service implements a single synchronous, message-passing operation between the tenancy admin and the A&B service whose operations rely on Cloud-based mechanisms customized for the MIDAS requirement to provide Cloud resource usages and costs for a tenancy.

3.2 END USER ANCILLARY USE CASES DIAGRAMS

The end user ancillary use cases diagrams are grouped in two categories according to the specification reported in D2.2: the use cases dealing with file management, and the use cases dealing with the test methods inspection. Both of them rely on the I&A mechanisms to allow verification of end user credentials. All file management use cases are single synchronous, message-passing operations that can be easily understood with the basic knowledge of any file system architecture. For any further detail please refer to the specifications reported in D2.2.

The test method inspection use cases, reported in Figure 3.4, give a presentation of the use (including the rationale of that use) but not the internal architecture of the test method. They explain what a test method does, why, when and where, for what it is useful, what kind of information the end user must supply to the test method, what kind of information the test task report says to the user. For the core services only, they are also exploited internally to obtain information about the test method properties, in particular:

- how to reach the MIDAS services involved in the execution of a specific test method by using some form of web service addressing;
- how to exchange data among instance of different MIDAS services during the execution of a test method. Currently, we expect two core methods for dealing with data exchange among service instance: using shared files on the shared tenancy file system and exchanging customized messages using a domain-specific language shared by the communicating service instances.
Figure 3.4: Test Method Inspection sequence diagrams

Also the method inspection operations are single synchronous, message-passing operations, and they are backed up in some implementation-dependent database responsible to provide reliable storage options to the implementation of the service.

3.3 END USER MAIN USE CASES DIAGRAMS

The sequence diagrams of the three main MIDAS end user use cases, i.e. direct test execution, manual test design and automated test design of Figure 2.2, are presented and discussed. These sequence diagrams depict the workflows among the end user and core services of the MIDAS platform for each use case. They do not depend on any specific test method, but are necessary to understand the interactions among the deployed MIDAS services, in particular when they will be deployed on Cloud.

3.3.1 Direct Test Execution Sequence Diagram

The goal of this use case is to execute a MIDAS-compatible TTCN-3 test suite. MIDAS allows access to users that are able to write TTCN-3 code, i.e., there are test methods that accept as input TTCN-3 code. In order to execute a MIDAS-compatible TTCN-3 test suite, end user and core services have to be orchestrated according to the interactions reported in the sequence diagram in Figure 3.5.

The steps of the sequence diagram are:

1. The end user (mediated, if necessary, by the MIDAS gateway/portal) uploads to the tenancy file system a file *fid1* containing MIDAS-compliant TTCN-3 code.
2. A success/failure response is returned to the end user.
3. The end user (mediated, if necessary, by the MIDAS gateway/portal) invokes the Test Gen&Run Service to request the execution of test method *id*, using as input the file *fid1* and with additional information encoded as *meta* objects.
Figure 3.5: Direct Test Execution sequence diagram
4. A success/failure response is returned to the end user containing the test method request task_id identifying its request, that will be used (step 27) to poll the status of the execution request.

5. The Test Gen&Run Service invokes the Test Method Query Service on test method id properties.

6. A success/failure response is returned to the Test Gen&Run Service containing, among other information, the web service endpoint wse1 of the Run Manager Service for test method id.

7. The Test Gen&Run Service invokes the Run Manager Service using wse1 to contact it, to request the execution of a run instance of task_id, using as input the file fid1, with additional information encoded as meta objects.

8. A success/failure response is returned to the Test Gen&Run Service containing the run_id identifying its test run instance of task_id, currently in the system that will be used to track the status of the running instance request (step 25).

9. The Run Manager Service interrogates the Test Method Query Service on test method id properties. The method identifier id is contained in the test method request task_id.

10. A success/failure response is returned to the Run Manager Service containing, among other information, the web service endpoint wse2 of the Executor Service.

11. If necessary, the Run Manager Service invokes the Executor Service using wse2 to perform test initialization procedures of the test method request task_id of the test method id, using as input the file fid1 (if interactions with the File Management Service is required), and with additional information encoded as meta objects.

12. If necessary, a success/failure response is returned to the Run Manager Service containing the init_id identifying its test initialisation request, currently in the system that will be used to track the status of the initialization request.

13. The Executor Service performs initialization.

14. The Executor Service invokes the Run Manager Service to communicate that the initialization procedure identified by init_id for the test method request task_id is done.

15. A response is returned to the Executor Service.

16. The Run Manager Service invokes the Executor Service using wse2 to perform test execution of the test method request task_id of the test method id, initialized by the procedure identified by init_id and using as input the file fid1, and with additional information encoded as meta objects.

17. A success/failure response is returned to the Executor Service containing the exec_id identifying its test execution request, referring to the corresponding
task_id, currently in the system that will be used to track the status of the execution request.

18.-22. The Executor Service performs execution, using fid1 as input and producing fid2 as output.

23. The Executor Service invokes the Run Manager Service to communicate that the execution of the procedure exec_id, initialized by init_id, for the test method request task_id, is done, and the results are store in the file fid2.

24. A response is returned to the Executor Service.

25. The Run Manager Service invokes the Test Gen&Run Service to communicate that the run of the instance run_id of the test method id, identified internally by task_id, is done, and the results are stored in the file fid2.

26. A response is returned to the Run Manager Service.

27. The end user (mediated, if necessary, by the MIDAS gateway/portal) polls for the test status and outcomes of the test method id, identified internally by task_id.

28. The Test Gen&Run Service returns to the end user the status and the outcome file fid2 of the request identified internally by task_id.

Note that in this use case (and in the following ones) we assumed that any data exchange among web services would be performed through the shared file system. When the size of the exchanged payload is small, it could also be possible to encode the information to be exchanged in SOAP messages, so reducing the number of interactions with the File Management Service.

Particular care must be taken in the Executor Service implementation on the Cloud, as two subsequent invocations, for initialisation and execution, cannot be assumed to be received by the same running instance of the Executor Service. Two different copies of the same instance could be contacted, due to failure or automatic load balancing. A simple solution consists in using the shared file system to synchronise, allowing the two different Executor Service instances to communicate (not shown in the sequence diagram).

### 3.3.2 Manual Test Design Sequence Diagram

The goal of this use case is to execute test cases and data that are provided in a non-executable and platform independent model. The test cases and data representation (and the target SAUT) must be MIDAS-compatible. This MIDAS-compatible test model is first processed to generate executable representations of the test cases and data, and then it is directly executed as in the Direct Test Execution use case (see Figure 3.6).

The steps of the sequence diagram are:

1. The end user (mediated, if necessary, by the MIDAS gateway/portal) uploads to the tenancy file system a file fid1 containing a test suite as UML sequence diagrams (MIDAS DSL).

2. A response is returned to the end user.
Figure 3.6: Manual Test Design sequence diagram
3. The end user (mediated, if necessary, by the MIDAS gateway/portal) invokes the Test Gen&Run Service to request the execution of **test method id**, using as input the file \(fid1\), and with additional information encoded as **meta** objects.

4. A success/failure response is returned to the end user containing the **test method request task_id** identifying its request, currently in the system that will be used (step 17) to poll the status of the execution request.

5. The Test Gen&Run Service invokes the Test Method Query Service on test method **id** properties.

6. A success/failure response is returned to the Test Gen&Run Service containing, among other information, the web service endpoint \(wse3\) of the Test Gen Service for test method **id**.

7. The Test Gen&Run Service invokes the Test Gen Service (e.g., for Model-to-TTCN-3 transformation) using \(wse3\) to contact it, to request the execution of a gen instance of **task_id**, using as input the file \(fid1\), and with additional information encoded as **meta** objects.

8. A success/failure response is returned to the Test Gen&Run Service containing the **gen_id** identifying its **test gen instance** of **task_id**, currently in the system that will be used to track the status of the running instance request (step 14).

9.-13. The Test Gen Service performs test generation, using \(fid1\) as input and producing \(fid2\) as output.

14. The Test Gen Service invokes the Test Gen&Run Service to communicate that the test gen instance **gen_id** of the test method **id**, identified internally by **task_id**, is done, and the results are stored in the file \(fid2\).

15. A success/failure response is returned to the Test Gen Service.

16. The Direct Test Execution is performed, using test method request **task_id**; the file \(fid2\) will be used as input, and the file \(fid3\) will be generated as output.

17. The end user (mediated, if necessary, by the MIDAS gateway/portal) actively asks for the test status and outcomes of the test method **id**, identified internally by **task_id**.

18. The Test Gen&Run Service returns to the end user the status and the outcome file \(fid3\) of the test method request identified internally by **task_id**.

### 3.3.3 Automated Test Design Sequence Diagram

The goal of this use case is to automatically generate test cases and data, and to execute them on the MIDAS platform. The end user supplies models for test generation (e.g., structural, functional, behavioural models), and the MIDAS platform firstly generates test cases and data in a non-executable platform independent model format. Secondly, this output is processed as in the previous use case. The steps of this use case are, with the adequate adjustments, identical to the Manual Test Generation use case (see Figure 3.6).
Figure 3.7: Automated Test Design sequence diagram
4 Tenancy Admin APIs

In this section, we specify the MIDAS Tenancy Admin services. These services are able to support all the tenancy admin use cases presented in Section 3.1.

4.1 Identification & Authentication Service

The identification and authentication operations enable the tenancy admins to create, list and delete end users in their tenancies. Only tenancy admins can access these operations.

There is an additional operation provided by this service, visible only to the MIDAS end user services and core services. This operation allows a MIDAS service to verify if a given user (or admin) exists in the tenancy and if an additionally submitted password is correct.

![Diagram of Identification and Authentication Service](image)

Figure 4.1 - Identification & Authentication Service: Interface and Data Types

MIDAS • Project Number 318786 • D6.2 • Page 31 of 52
### 4.1.1 Create User

The operation allows the creation of a new end user in a tenancy.

<table>
<thead>
<tr>
<th>Operation</th>
<th>createuser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>synchronous request/response</td>
</tr>
<tr>
<td>Description</td>
<td>Create a new end user in the tenancy. The new end user can access, after login, all end user services.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>CreateUserIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>userFullname</td>
<td>String</td>
</tr>
<tr>
<td>userEmail</td>
<td>String</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
</tr>
<tr>
<td>pwd</td>
<td>String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>CreateUserOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>SynStatus ; done</td>
</tr>
<tr>
<td>declineMsg</td>
<td>String[0..1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>Input is well-formed.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Post-conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>status = done</td>
</tr>
<tr>
<td>If no end user with the given username exists, a new end user with the given information is created and permanently stored - declineMsg is null.</td>
</tr>
<tr>
<td>status = decline</td>
</tr>
<tr>
<td>At least one of the pre-conditions is not fulfilled - declineMsg documents the pre-condition violations.</td>
</tr>
<tr>
<td>status = drop</td>
</tr>
<tr>
<td>If an end user with the given username exists, no action is performed - declineMsg is null.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>status = fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>The operation cannot be carried out because of internal failure -</td>
<td></td>
</tr>
</tbody>
</table>
4.1.2 Delete User

The operation allows the deletion of an existing end user in a tenancy.

<table>
<thead>
<tr>
<th>Operation</th>
<th>deleteUser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>synchronous request/response</td>
</tr>
<tr>
<td>Description</td>
<td>Delete an existing end user from the tenancy, if it exists.</td>
</tr>
<tr>
<td>Input</td>
<td>DeleteUserIn</td>
</tr>
<tr>
<td></td>
<td>username: String</td>
</tr>
</tbody>
</table>

| Output | DeleteUserOut |
| | status: SynStatus; done | decline | drop | fail |
| | declineMsg: String [0..1] |

| Pre-conditions | ● Input is well-formed. |
| Post-conditions | ● status = done |
| | If an end user with the given username exists, he is permanently deleted - declineMsg is null. |
| | ● status = decline |
| | At least one of the pre-conditions is not fulfilled - declineMsg documents the pre-condition violations. |
| | ● status = drop |
| | No end user matching username in the tenancy - declineMsg is null. |
| Exceptions | ● status = fail |
| | The operation cannot be carried out because of internal failure - declineMsg is null. |
4.1.3 List Users

The operation allows retrieving the users of a tenancy.

<table>
<thead>
<tr>
<th>Operation</th>
<th>listUsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>synchronous request/response</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the list of users of a tenancy with their full names, user names, emails and if they are tenancy administrator or not.</td>
</tr>
<tr>
<td>Input</td>
<td>None</td>
</tr>
<tr>
<td>Output</td>
<td>ListUsersOut</td>
</tr>
<tr>
<td>status: BasicStatus ; done</td>
<td>drop</td>
</tr>
<tr>
<td>userInfos : (UserInfo)</td>
<td></td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>None</td>
</tr>
</tbody>
</table>
| Post-conditions | ● status = done  
The userInfos data structure reports the list of users in the tenancy.  
● status = drop  
The operation cannot be carried out because of resource unavailability. |
| Exceptions | ● status = fail  
The operation cannot be carried out because of internal failure. |

4.1.4 Verify User

The operation allows the verification of provided username and password information w.r.t. a tenancy persistently stored information pertaining the same username.

| Operation | verifyUser |
| Interaction | synchronous request/response |
| Description | Verify if a given user with a given password exists in the tenancy. |
### Input

**VerifyUserIn**
- `username`: String
- `pwd`: String

### Output

**VerifyUserOut**
- `status`: `SynStatus`; `done`, `decline`, `drop`, `fail`
- `userInfo`: `UserInfo`
- `declineMsg`: String [0..1]

### Pre-conditions

- Input is well-formed;

### Post-conditions

- `status = done`
  If an end user with the given `username` and the given `password` exists, `declineMsg` is null.
- `status = decline`
  At least one of the pre-conditions is not fulfilled - `declineMsg` documents the pre-condition violations.
- `status = drop`
  If no end user with the given `username` exists or the given `password` is incorrect, `declineMsg` is null.

### Exceptions

- `status = fail`
  The operation cannot be carried out because of internal failure - `declineMsg` is null.

---

#### 4.2 Accounting & Billing Service

The account and billing service provides to the tenancy admin the information about the current costs incurred while exploiting the MIDAS resources (Cloud resource and test method portfolio) involved in the provisioning of the End User Services, Core Services and the Tenancy Admin Services.

Since the concept of user identity is not transferred between end user services and core services, it is impossible to track down at the level of core services who is invoking the APIs. At the same time, it is not possible to track the time spent invoking end user services since all core services are asynchronous, and the time spent between an end user service invocation and the completion of the requested activity can be impacted by the local scheduling mechanisms of the core services containers. Moreover, since there is no concept of file ownership in the tenancy file system, it is impossible to track down the usage of user files inside a tenancy.
Given these premises, the accounting and billing service can provide to the tenancy admin just the current costs incurred while the tenancy end users exploit the MIDAS resources involved in the provisioning of all tenancy services, without any further finer grain.

![Figure 4.2 - Accounting & Billing Service Interface and Data Types](image)

### 4.2.1 Get Billing Information

<table>
<thead>
<tr>
<th>Operation</th>
<th>getBilling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>synchronous request/response</td>
</tr>
<tr>
<td>Description</td>
<td>Given a specific month and year, it returns a CSV file containing the billing information for the specified period. During the current billing period (monthly), the CSV file contains an estimate of the costs that may change throughout the billing period until a final consolidated report is generated at the end of the billing period.</td>
</tr>
<tr>
<td>Input</td>
<td>GetBillingIn</td>
</tr>
<tr>
<td></td>
<td>month: Month</td>
</tr>
<tr>
<td></td>
<td>year: Year</td>
</tr>
<tr>
<td>Output</td>
<td>GetBillingOut</td>
</tr>
<tr>
<td></td>
<td>status: SynStatus ; done</td>
</tr>
<tr>
<td></td>
<td>declineMsg: String [0..1]</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Post-conditions</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| file: CsvFile [0..1] | ● Input is well-formed;  
● Month value is between 1 and 12 (inclusive);  
● Year is lower than or equal to current year;  
● Year is greater than or equal to the tenancy registration year. | ● status = done  
A CSV file is returned, containing the billing information for the requested month and year - `declineMsg` is null.  
● status = decline  
At least one of the pre-conditions is not fulfilled - `CsvFile` is null and `declineMsg` documents the pre-condition violations. | ● status = fail  
The operation cannot be carried out because of internal failure - `CsvFile` is null and `declineMsg` is null. |
5 MIDAS PLATFORM ON THE CLOUD

The first part of this section gives an overview of the current solutions provided by Amazon Web Services (AWS) Cloud for the design and deployment of SOA applications on a Cloud infrastructure. In particular the analysed solutions concern the design and blending of application building blocks, the possibility to achieve computing elasticity, and to tackle data storage reliability, persistency and fault tolerance.

In the second part of this section, four Cloud resource scalability dimensions are identified based on the analysis of the MIDAS requirements (more specifically general and functional requirements), of the MIDAS architecture and specifications as they are discussed in D2.2. Each scalability dimension is discussed with reference to the proposed MIDAS deployment strategy, and the Amazon Cloud solutions to implement each dimension and to meet the MIDAS requirements and specifications are identified.

5.1 AMAZON CLOUD SOLUTIONS FOR MIDAS

The Amazon Web Services (AWS) Cloud provides a highly reliable and scalable infrastructure for deploying web-scale solutions, with minimal support and administration costs, and more flexibility than an in-house infrastructure, either on-premise or at a datacentre facility [6]. AWS offers a large ecosystem of infrastructure services focused to help developers to design, deploy and operate their web-scale applications on the Cloud.

5.1.1 Cloud/Computing Building blocks

The core building block of the Amazon Cloud is the Amazon Elastic Compute Cloud (Amazon EC2) service. Amazon EC2 is a web service that provides resizable computing capacity in the Cloud.

It is possible to bundle an operating system, application software and associated configuration settings into a specialised Virtual Machine Image (VMI) called Amazon Machine Image (AMI). The VMI/AMI is the basic deployment unit of services and applications on the Amazon Cloud, and a single AMI is used to provision multiple virtualized Amazon EC2 instances as well as decommission them using simple web service calls to scale capacity up and down quickly, as the application’s capacity requirement changes.

While AMIs are basic unit of deployments, Amazon EC2 instances are the fundamental building block for building a required computing infrastructure in the AWS Cloud. Instances can be thought as virtual servers that can run applications. To create an infrastructure from EC2 resources, the user first requires the launch of one or several instances, for which he specifies the instance type and the AMI; the user can specify any AMI previously registered with Amazon, including Amazon’s or the user’s own. Once the AMI has been transparently deployed on a physical machine (the resource status is running), the instance is booted; at the end of the boot process the resource status becomes installed. The installed resource can be used as a regular computing node immediately after the booting process has finished, via an ssh connection.
The Amazon EC2 does not provide job execution or resource management services; a Cloud resource management system can act as middleware between the user and Amazon EC2 to reduce resource complexity. Amazon EC2 abides by a Service Level Agreement in which the user is compensated if the resources are not available for acquisition at least 99.95% of the time, 365 days/year.

5.1.2 Elastic Computing Service

Amazon CloudWatch provides a reliable, scalable, and flexible monitoring solution for AWS cloud resources. For Amazon EC2 instances, Amazon CloudWatch collects and reports metrics for CPU utilization, data transfer, and disk usage activity from each Amazon EC2 instance at a one-minute and five-minute frequency.

Amazon Auto Scaling allows to scale Amazon EC2 capacity up or down automatically according to some user-defined conditions. With Auto Scaling, it is possible to ensure that the number of Amazon EC2 instances used increases seamlessly during demand spikes to maintain performance, and decreases automatically during low demand to minimize costs.

Auto Scaling is enabled by Amazon CloudWatch solution. CloudWatch is used to monitor Amazon EC2 instances, and when certain configurable events happen, Amazon Auto Scaling can launch more instances based on any pre-defined AMI template or de-activate some idle instances previously launched. When that more instances of a pre-defined AMI template are running, Amazon Elastic Load Balancing (Amazon ELB) masks these instances and distribute incoming traffic in a round-robin pattern between all the instances assigned to it, and it makes possible to control in real time how many instances to launch to cover sporadic bursts of high-volume traffic, and keep at least one or two running during traffic lulls. If any of the EC2 instances fails to respond, Amazon ELB will detect it and launch a replacement. When web traffic dies down, you can terminate instances automatically, too.

5.1.3 Storage Services

AWS offers multiple Cloud-based storage options. Each has a unique combination of performance, durability, availability, cost, and interface, as well as other characteristics such as scalability and elasticity. These additional characteristics are critical for web-scale Cloud-based solutions [7].

Amazon EC2 instance store volumes (also called ephemeral drives) provide temporary block-level storage for many Amazon EC2 instance types. This storage consists of a preconfigured and pre-attached block of disk storage on the same physical server that hosts the Amazon EC2 instance. The amount of this disk storage varies by Amazon EC2 instance type. In those Amazon EC2 instance families that provide instance storage, larger instances tend to provide both more and larger instance store volumes. Note that some instance types, such as the micro instances, have no instance storage provided. In general, local instance store volumes are ideal for temporary storage of information that is continually changing, such as buffers,
caches, scratch data, and other temporary content, or for data that is replicated across a fleet of instances, such as a load-balanced pool of web servers.

Amazon EC2 instance storage is a well-suited solution for this purpose. It consists of the virtual machine’s boot device, plus one or more additional volumes that are dedicated to the Amazon EC2 instance. This storage is usable only from a single Amazon EC2 instance during its lifetime. Note that instance store volumes cannot be detached or attached to another instance.

Amazon Elastic Block Store (Amazon EBS) volumes provide durable block-level storage for use with Amazon EC2 instances. Amazon EBS volumes are off-instance, network-attached storage that persists independently from the running life of a single Amazon EC2 instance. After an Amazon EBS volume is attached to an instance, it can be used as a physical hard drive, typically by formatting it with the file system of choice and using the file I/O interface provided by the instance operating system. An Amazon EBS volume can be used to boot an Amazon EC2 instance, and multiple Amazon EBS volumes can be attached to a single Amazon EC2 instance. Note, however, that any single Amazon EBS volume may be attached to only one Amazon EC2 instance at any point in time.

Amazon Simple Storage Service (Amazon S3) is a simple storage service that offers software developers a highly-scalable, reliable, and low-latency data storage infrastructure at very low costs. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from within Amazon EC2. It is possible to write, read, and delete objects containing from 1 byte to 5 terabytes of data each. The number of objects you can store in an Amazon S3 bucket is virtually unlimited. Amazon S3 is also highly secure, supporting data encryption, and providing multiple mechanisms that allow fine-grained control of access to Amazon S3 resources. Amazon S3 is also highly scalable, allowing concurrent read or write access to Amazon S3 data by many separate clients or application threads. Amazon S3 automatically manages scaling and distributing redundant copies of your information to other servers in other locations in the same region, all using Amazon’s high-performance infrastructure. Amazon S3 is optimal for storing numerous classes of information that are relatively static and benefit from its durability, availability, and elasticity features.

Amazon Relational Database Service (Amazon RDS) is a web service that provides the capabilities of MySQL, Oracle, or Microsoft SQL Server relational database as a managed, Cloud-based service. It also eliminates much of the administrative overhead associated with launching, managing, and scaling a proprietary relational database on Amazon EC2 or in another computing environment.

5.2 MIDAS DEPLOYMENT ON AMAZON CLOUD

In Figure 5.1 the overall picture of MIDAS TaaS deployment approach on the Cloud is drawn. Even though labs are not included in the basic MIDAS TaaS platform, in the Figure both Labs and Tenancies are reported to show their symmetric behaviour from the Cloud deployment point of view.
Four main Cloud resource scalability dimensions are sketched:

**Tenancy/lab Space Scalability** - Cloud resources are allocated to a tenancy/lab upon its creation, according to the agreement established with the MIDAS TaaS admin. In fact, the amount of allocated Cloud resources is agreed according to a SLA between the MIDAS TaaS admin and the customer Tenancy/Lab administrator. Each time a new tenancy is created/allocated a new pool of Cloud resources should be assigned to it. Accordingly, upon tenancy/lab deletion all the currently associated Cloud resource space has to be removed, that is all VMs hosting computing services have to be detached from the pool of allocated VMs to be available as free resources for MIDAS TaaS.

**Computing elasticity** - inside a Tenancy/Lab space, initially allocated resources can scale up and down according to the current usage of the tenancy computing resources: when new computing resources are necessary, either because the number of registered users in a tenancy increases, or because end users testing activities amount increase, the Cloud autoscaling facilities should allocate new computing resources, i.e. VMs, hosting the same core and file management services, and redirect and schedule service invocations from users on one of the available VMs.

**Storage Scalability/Redundancy** - persistent storage for end users data (like, test logs, journals, models, etc.) is designed to be physically shared, although logically separated (sandboxed) among different tenancy/lab spaces, that means among users belonging to different tenancies. Therefore, the Cloud infrastructure has to provide a global storage for MIDAS and mechanisms to ensure insulations/protection of data among different tenancies. Although we figure out a single storage entity, the Cloud infrastructure should offer autoscaling, as well as redundancy of data storage to cope with failures.

**Portfolio Redundancy** - the MIDAS portfolio is the repository of Test Methods and Test Components developed by Lab users and used by Tenancy users to carry out their testing activities. As such, the portfolio storage is persistent as well as physically and logically shared among all MIDAS users. While tenancy users access the Portfolio in read-only mode, Lab users, that are Test Methods developers, have read/write access to the portfolio\(^1\). In the deployment design we will rely only on Cloud storage redundancy mechanisms in order to provide the MIDAS portfolio with the required fault tolerance features.

\(^1\) It is out of scope of this document to discuss the visibility of Portfolio data to different users profiles. At the current stage it is assumed that the MIDAS TaaS admin manages the portfolio making published methods (and components) visible to tenancy end users.
5.2.1 MIDAS Building Blocks

The MIDAS platform is designed as a Service Oriented Architecture that can fully leverage the Web services paradigm to develop, deploy and operate on a Cloud. The MIDAS services do not have tight dependencies on each other, so each service can be developed, deployed and managed by the Cloud independently from the others.

In theory, each MIDAS service could have its own VMI customised with the implementation-dependent software requirements (libraries, runtime environments, databases, ancillary services) as a basic deployment unit for the specific service implementation. All core services in MIDAS communicate asynchronously with the others and treat them as “black boxes”, without code or functional dependencies. With reference to the MIDAS deployment strategy depicted in Figure 5.1, the MIDAS TaaS is composed of Tenancy/Lab instances. Each Tenancy/Lab instance is logically...
composed of two basic deployment units, i.e., VMIs, one for tenancy/lab administration and one for end user/core services. In the proposed deployment strategy, end user services, i.e. Test Gen&Run, File Management and Test Method Query services, are hosted in the same deployment unit, together with specific core services implementations, while the Identity & Authentication service is hosted on a different deployment unit together with the Accounting & Billing service. The rationale of this choice is due to the different scalability and elasticity requirements of the two groups of services. Each Tenancy/Lab instance has its own private Cloud resources pool that can scale by exploiting the elasticity services of the underlying Amazon Cloud infrastructure. In particular, a pool of VMIs should be allocated to each new Tenancy/Lab instance according to the agreement/contract with the TaaS admin.

Since each Tenancy/Lab end user service can live its own “life” independently from the others, in principle there is no functional requirement that obliges to make two end user/core services live in the same deployment unit. The rationale of grouping Tenancy/Lab end users and core services in a single deployment unit is twofold: 1) the entire pool of end user services could be scaled as a whole through the Amazon elasticity mechanisms, 2) for the time being, there are no specific requirements to partition and distribute end user services on separate deployment units. On the other side, I&A and A&B services implement administration tasks that are worth to be centralized and resident on one computing/data instance. Nevertheless, Amazon automatically manages redundancy of unit instances to deal with failures.

In order to drive the implementation of the MIDAS end user and core services according to the proposed deployment strategy, we designed a development environment for the MIDAS project made available to all MIDAS partners. The proposed environment enables them to implement, debug and test their services implementations on a seamless and loosely coupled platform. The environment is based on three open source technologies: Oracle VM VirtualBox, Vagrant and Ansible. Vagrant is a free software tool for creating customizable, lightweight, reproducible, and portable development environments made up of VMIs. It relies on a Virtual Machine Monitor (VMM) whose task is to create and run VMIs. The default VMM for Vagrant is Oracle VM VirtualBox. Eventually, Ansible is a tool to automatically build and deploy on different virtual machine instances the VMIs designed by the MIDAS project, including Amazon EC2 instances and related web services. The complete description of this development environment is provided in the Annex A, together with a companion user guide.

### 5.2.2 MIDAS Computing Elasticity

The service architecture of the MIDAS platform is naturally suitable for automatic elasticity. The statelessness and loosely coupling of the MIDAS services represent the key enabling factors to fully provide automatic reconfiguration of MIDAS components depending on their usage.

To fully implement automatic elasticity mechanisms in a Cloud-based, service-oriented application, three key ingredients are necessary:
1. the Cloud infrastructure must provide some mechanisms to monitor the resources usage of the applications,

2. the Cloud infrastructure must provide mechanisms to automatically scale up and down the virtual machines hosting running instances of the same service container of a Tenancy/Lab, and

3. the virtual machines allocated to the same Tenancy/Lab should be managed by a load balancer able to steer and schedule connection requests to the instances according to some predefined algorithm.

As previously described, Amazon provides services to address all the aforementioned issues. Indeed, Amazon CloudWatch allows monitoring of Amazon EC2 instances, and when certain configurable events happen, they are used to trigger Amazon Auto Scaling to launch more EC2 instances as replicas of the existing ones. All replicas of one EC2 instance can be identified as such since they can only run on a pre-defined AMI template, Amazon Elastic Load Balancing (Amazon ELB) identifies EC2 instance replicas and can distribute incoming traffic in a round-robin pattern between all them.

5.2.3 MIDAS Storage Scalability and Redundancy

The MIDAS platform requirements on the storage capabilities are three-fold:

1. MIDAS services need temporary disk space for implementation-dependent service execution; each single running service may temporally write/read data whose persistence lasts from the invocation time to the reply time.

2. MIDAS needs a short-term persistent disk space to implement custom data sharing among services during single test method generation and run activities; end user core services, like test running, are composite service orchestrating more atomic services. Although all MIDAS services are stateless, component services in an orchestration may communicate exchanging data files through a shared memory disk space. In that case, the persistence of these data must last from the invocation time to the reply time of the composite service.

3. MIDAS must supply a persistence space for user data including models, test data, test logs, journals and documentation.

While the Amazon EC2 instance store volumes provide a satisfactory solution for the temporary and short-term persistent disk spaces (i.e., for the storage requirements 1 and 2), for the user data persistent space (i.e., for the storage requirement 3), a suitable solution is to rely on Amazon S3, instead of Amazon EBS volumes since they do not allow to share data among MIDAS components running on different Amazon EC2 instances. In addition, the Amazon S3 data model can be used to logically partition in a sandboxed way data among different Tenancy/Lab users. Amazon S3 autoscaling and redundancy features would allow to meet the MIDAS requirements concerning data scaling and data replication for fault tolerance. Eventually, Amazon RDS is not
necessary for the MIDAS architecture “per se”, but, since the implementation of the end user, core and tenancy admin services could require to store structured information with frequent accesses, a database solution as an automatically managed Cloud-based service will be useful.

5.2.4 Preliminary Cloud Cost Analysis

In this section, we perform a preliminary analysis of the Cloud costs that MIDAS will incur until the end of the project, plus six months considered for exploitation, i.e. for the next 24 months. The analysis is carried out taking into account the Amazon Web Services we plan to use, some estimated scenarios on their usage, and the budget constraints of the project.

The most expensive Amazon services we plan to use are:

- the EC2 service, to deploy the MIDAS platform on the Amazon VMs,
- the S3 scalable storage service, to provide a permanent storage solution to tenancies,
- the CloudWatch service, to perform resource and application monitoring.

Starting from these services and taking into account the requirements reported in D6.1 [1], the MIDAS architecture on Cloud presented previously, and the principles of economy, we envision the following potential scenarios for what concerns Cloud resources:

1. A single VM of type m1.small (0.065 $ per hour) to host the tenancy admin services (Identity & Authentication and Accounting & Billing), and a single VM of type m1.xlarge (0.52 $ per hour) to host the end user and core services for a single tenancy.

2. A single VM of type m1.small (0.065 $ per hour) to host the tenancy admin services (Identity & Authentication and Accounting & Billing), a single VM of type m1.small (0.065 $ per hour) to host test generation services (together with end user services) and a single VM of type m1.xlarge (0.52 $ per hour) to host the test execution services.

The first scenario refers to the MIDAS deployment strategy proposed in this document (and depicted in Figure 5.1) that is the one currently adopted and agreed for the project. The second scenario could be considered in successive increments of the MIDAS prototype if new requirements (e.g. test generation and execution might have very different resource demands) will arise later on during the project lifetime.

At the current stage of the project, we do not see any feasibility reason to use more VMs for the minimal deployment of a single tenancy. Of course, during the lifecycle of the tenancy, more VMs could be used, depending on the computing elasticity required by the tenancy end user.

According to the first scenario, the resource allocation policy (how the MIDAS TaaS admin pays the Cloud provider) for a single tenancy T, depending on the contract between the tenancy admin and the TaaS admin, could specify a maximum number
1+N_i of VMs to instantiate during the lifetime of a tenancy, resulting in a (maximum) allocation of m1.small + N_1 * m1.xlarge types of instances. Analogously, we could estimate a maximum allocation of m1.small + N_i * m1.small + M_i * m1.xlarge for the second scenario, where N_i and M_i quantify the independent elasticity requirements for test generation and test execution.

To compute the final Amazon WS Cloud cost estimate for a single tenancy, we must separately add the fixed costs for:

- Storage on S3: 100 GB with 10,000 requests (GET, PUT, POST) cost 9.58 $ per month;
- Elastic Load Balancing: 1 load balancer costs 20.58 $ per month;
- CloudWatch: depending on metrics, it costs at max 3.50 $ per month;
- Elastic IP: 3 elastic IPs cost 11.31 $ per month;
- Amazon RDS: 5GB t1.small MySQL instance costs 26.17 $ per month;
- SAUT communication (EC2 to internet): 10 GB transfer costs 1.08 $ per month.

Assuming 8 hours per day as upper bound for the Amazon Cloud usage by the whole project, the monthly costs *per tenancy* of the Cloud for the first scenario are about 88 $ + 125 $ * N_i.

For the considered 24 months, we will assume to have a single “development” tenancy operational 8 hours per day with N_i = 2, for a total cost of 8,000$ and for the last 12 months of the project, we assume to have two “pilot” tenancies operational 8 hours per day with N_i = 2, resulting in a predicted total cost of Cloud resources for the whole project of 16,000$, that is within the limits of the 15,000€ requested as contribution for Cloud resources.
6 CONCLUSIONS

This document provides a description of the MIDAS Testing As A Service platform starting with a systematic description of the classes of its users and their interactions with the platform components, in terms of sequence diagrams. This step allowed to identify the building blocks of the platform from the Cloud point of view, i.e. for its deployment on the Cloud.

The set of additional MIDAS services necessary to interface the underlying Cloud infrastructure, together with their defined APIs, are depicted in relation to the identified classes of users.

Then, the strategy adopted to deploy the MIDAS TaaS on the chosen public Cloud IaaS is introduced by detailing first the services provided by the selected Cloud provider necessary to provide the main functionalities related to the Cloud deployment of MIDAS. Hence, the solutions adopted to deploy and integrate the MIDAS building blocks on the Cloud are described including multi-tenancy and Cloud storage facilities. The strategies to tackle the elasticity and scalability requirements of the MIDAS platform are detailed.

Finally, a preliminary analysis of the costs deriving from the usage of Cloud resources is reported, even though it needs to be further detailed once the Cloud resource requirements of the MIDAS pilots will be more specific.

In addition, the document provides in appendix the description of the preliminary basic prototype for the MIDAS TaaS platform, to show a first deployment on the Cloud according to a monolithic scenario where all services are deployed in one Virtual Machine.

As mentioned in the document, some aspects related to the usage of the underlying Cloud infrastructure will not be addressed within the MIDAS project since they regard the industrialization phase of the MIDAS TaaS platform for future exploitation. This is the case of the Service Level Agreements that are necessary for an IaaS deployment to establish specific terms and conditions to be negotiated with the Cloud provider for the availability and performance of the servers, network, and storage infrastructure, together with the specification of procedures for the maintenance and management of the Cloud infrastructure and how any potential downtime is handled.
A Basic Architecture For the Integrated MIDAS Testing Platform on the Cloud.

This appendix describes the architecture deployed on the Cloud for the basic integrated environment of the MIDAS SOA Testing Platform.

In the previous deliverable D6.1, we showed some preliminary experiences in installing and configuring the TTworkbench Software tool on a virtual machine configured on a basic Cloud infrastructure. Starting from that, as additional work, we have put into operation a real integrated platform able to execute TTCN-3 test cases against a real SUT. More specifically, taking advantage of the consolidated software tools such as Vagrant and Ansible, we have built a first monolithic deployment scenario for the MIDAS SOA testing platform as service. That monolithic scenario consists of one virtual machine able to run on the Cloud and to host all web services developed by the MIDAS partners in order to put into operation the SOA architecture of the MIDAS platform.

As specific work, we implemented and deployed on this virtual environment the implementations of the MIDAS End User and Core services described in Section 2.3. In particular, the implemented services are the ones related to the Direct Test Execution use case, as detailed in the Section 3.3.1, that are deployed using Apache Tomcat service container, as shown in Figure A.1.

![Diagram of the monolithic container](image)

Figure A.1: The monolithic container that encloses all MIDAS web services

---

2 These software tools are described in the document “MIDAS_Virtual_Machine_Image_Guide_v1.2” that, as previously mentioned, was drafted to provide the MIDAS partners with an environment for the deployment of their services and test methods.
In more detail, for the Executor Service, we developed and deployed a wrapper in Java of the TTworkbench executor, i.e. the TTman component that allows us to run the already compiled TTCN-3 test campaign script.

In the remaining of this appendix, we describe the architecture of the integrated MIDAS SOA testing platform we put into operation in order to support a preliminary demo prototype related to the Direct Test Execution use case.

### A1 THE MIDAS SERVICE ORIENTED ARCHITECTURE INFRASTRUCTURE

In this preliminary basic prototype for the MIDAS SOA testing platform as a service, we configured a monolithic scenario described in D6.1 [1], in which all services are deployed in the same virtual machine. The Figure A.2 shows a high level picture of this scenario. More specifically, the Tenancy Admin Services, the End User Services and the Core Services are contained in only one virtual machine, deployed in an EC2 instance of Amazon AWS.

The virtual machine is composed of:

- **MIDAS Portal**: it is a web portal that allows end users to login in the MIDAS SOA testing platform, to manage the users in the tenancy (only the tenancy admin can access this utility), to access the storage service that provides users with a tool to upload their files (the TTCN-3 test campaign files), and to invoke the Direct Test Execution upon these files, and last, a web page that informs the user about the state of his/her submitted execution tasks.

- **MIDAS TaaS**: this is built on top of an Apache Tomcat service container in which all the MIDAS services are deployed. It also contains a web service wrapper (the TTCN-3 Executor Service) to the TTworkbench TTman software tool.

- **TTworkbench**: TTCN-3 software tool selected by the MIDAS Consortium to execute the TTCN-3 scripts.

- **DB I&AS**: it is the database containing the identifiers of the tenancy users managed by the Identity&Authentications Service.

- **DB Test Method**: it is the database containing the Test MethodId, managed by the TestMethodQueryService.

- **S3 Storage**: it contains all TTCN-3 test campaign files inserted by the MIDAS end users and test method developers.

The License Server is the license manager for the TTworkbench software tool that is installed on a server outside the Cloud.

The System Under Test is a server outside the Cloud, running an example of SUT for the Bank ATM service. The service accepts SOAP requests and it responds with SOAP messages to communicate the results of the test.
A2 THE DIRECT TEST EXECUTION EXAMPLE

To put into operation the MIDAS architecture described in the previous subsection, we set up the two servers outside the Cloud within the CNR premises, and the MIDAS SOA testing platform deployed in EC2 Amazon instance on the Amazon Cloud, containing both the MIDAS Portal and the MIDAS TaaS.

When the servers (the external servers and the VM) are running, it is possible to execute the Direct Test Execution use case by uploading the TTCN-3 Test Campaign examples provided by the MIDAS partners (i.e. the AccountMngtExample and the LoginExample). In particular, it is possible to execute the LoginExample, through the clf file “TestPackage_TestContext.clf”, that is a simple TTCN-3 test script for login testing, and the AccountMngtClientEmulator, through the clf file “AMClientEmulator.clf”, that emulates the testing of a debit call towards the ATM service of a bank. The latter TTCN-3 test script is a complete SOA testing scenario. Indeed, a MIDAS end user can login within the MIDAS Portal, upload the clf file, and invoke the Direct Test Execution use case that implements the workflow described in the Section 3.3.1, that is, the Direct Test Execution sequence diagram shown in Figure 3.5. More specifically, when
the MIDAS end user invokes the test, the MIDAS platform activates (calls) the TestGenAndRunService, and follows all the steps until reaching the Executor Service that invokes the TTworkbench TTman executor. In the AccountMngtClientEmulator, for instance, the TTman, after checking the license by communicating with the License Server, runs the clf file and sends SOAP requests towards the SUT ATM example. The SUT, once received the SOAP message, executes the request and sends a SOAP message to inform the Executor Service with the results of the test.
REFERENCES


[2] MIDAS Deliverable D2.1: “Requirements for automatically testable services and services architecture”


