Dipartimento di Informatica
Università del Piemonte Orientale “A. Avogadro”
Via Teresa Michel 11, 15100 Alessandria
http://www.di.unipmn.it

The TAAROA Project Specification

Authors: Cosimo Anglano, Massimo Canonico, Marco Guazzone and Matteo Zola
({cosimo.anglano,massimo.canonico,marco.guazzone,matteo.zola}@unipmn.it)

TECHNICAL REPORT TR-INF-2009-02-02-UNIPMN
(February 2009)

Recent Titles from the TR-INF-UNIPMN Technical Report Series

The TAAROA Project Specification [*]

Cosimo Anglano Massimo Canonico Marco Guazzone
Matteo Zola
Department of Computer Science, University of Piemonte Orientale, Alessandria (Italy),
email:{cosimo.anglano,massimo.canonico,marco.guazzone,matteo.zola}@unipmn.it

Abstract

Since its introduction, the Grid computing paradigm has been widely adopted both in scientific and also in industrial areas. The main advantage of the Grid computing paradigm is the ability to enable, in a transparent way, the sharing and the coordination of several heterogeneous and large-scale distributed resources belonging to different institutional domains. One of its limitation is the lack of facilities for executing services. In fact, Grid computing has been traditionally used and improved for running computational-intensive or data-intensive applications. A service differs from this kind of applications in that it usually waits for requests from clients and replies with useful information; moreover, a service is typically subjected to some predefined constraints, called Service Level Agreement (SLA), including both temporal and performance restrictions. In this paper we present the TAAROA middleware, a software system that tries to extend the traditional target of the Grid computing paradigm to include the service concept. It attempts to accomplish its goal by using the virtualization technology. By abstracting the hardware and software resources of a computer, virtualization brings to TAAROA two important benefits: (1) the encapsulation of the service runtime environment, and (2) the possibility, through the migration facility, to move a service from the computer where it is running to another one that hopefully reduces the risk of violating some of the SLA constraints. In the current version of TAAROA middleware there is no explicit mechanism for achieving the level of a service as defined by the related SLA; this means that actually TAAROA is only able to provide a best-effort service.

Keywords: Grid Computing, Service Level Agreement, Virtualization.

[*]This work has been supported by TOP-IX and the Piedmont Region Agency under the Innovation Development Program.

1 Introduction

The traditional use of a distributed computing infrastructure is for executing computational-intensive or data-intensive applications for solving complex problems. These applications are characterized by the lack of user interaction and by the high demand of computational power or storage capacity. While almost all of the scientific applications can be considered to belong to at least one of the above categories, there are others, like the ones in the business domain, that follow different behavioral patterns. Services are an example of such applications.

A service is an application that differs from traditional resource-intensive applications for at least two aspects: (1) there is some kind of interaction with its users, that is it spends most of its time waiting for client requests (generally issued by a user) and, upon a request arrival, replies to a requesting client with useful information, and (2) is typically subjected to predefined temporal, performance and economical constraints referred to as Service Level Agreement (SLA). The importance of services is demonstrated by the actual trend in the development and in the deployment of applications [1, 7]: the Service Oriented Architecture (SOA) model is now a fundamental part in designing and integrating applications since it allows existing IT infrastructure and systems to achieve end-to-end enterprise connectivity by removing redundancies, generating unified collaboration tools, and streamlining IT processes [4, 2].

Distributed computing paradigms still lacks of suitable mechanisms in order to execute this kind of applications. Specifically, the two most challenging and still open problems are: (1) to decide to what physical machine assign a service for execution in order to satisfy its SLA constraints and (2) to continuously monitor the execution of the already running services for preventing and possibly reacting to SLA violations.

Finding the optimal allocation of a certain number of services to a finite number of physical machine, subjected to SLA constraints, is a computationally hard problem (NP-complete) [5]. Furthermore, when no such optimal allocation can be found, a quasi-optimal allocation is still required in order to minimize SLA violations; in this case, additional issues must be taken into consideration for the conflicting nature of the problem: two or more services competes for getting assigned to the machine that allows them to meet the largest number of SLA constraints.

The other challenging problem is the monitoring and the fulfillment of SLA constraints. It consists in observing the behavior of the service execution, collecting the performance measures related to SLA constraints, predicting future behavior (on the basis of a specific performance model) and finally properly reacting in order to prevent SLA violations, without compromising the SLA constraints of other running services.

In this paper we present the TAAROA middleware, a software system that tries to extend the traditional target of the Grid computing paradigm to include the service concept. It attempts to accomplish its goal by using the virtualization technology. By abstracting the hardware and software resources of a computer, virtualization brings to TAAROA two important benefits: (1) the encapsulation of the service runtime environment, and (2) the opportunity, through the migration facility, of moving an executing service from the physical machine where it is running to another one which, hopefully, reduces the risk of violating some of the SLA constraints. Even if the ultimate goal of TAAROA middleware is to enable the optimal execution of services on distributed systems, in its current version there is no explicit mechanism for achieving the level of a service as defined by the related SLA; this means that actually TAAROA is only able to provide a best-effort service.

The rest of this paper is organized as follows. In §2 we aim to provide an understanding of the main ideas underlying the TAAROA middleware by outlining its high-level architecture. In §3 we add details to the architecture described in §2 by illustrating the role of each TAAROA component and the associated interactions. In §4 we describe what are the information maintained by the TAAROA middleware, to enable interaction between its component, and how they are organized. In §5 the communication protocol, used by TAAROA components for interacting, is thoroughly described. Finally, in §6 we provide conclusions and our future research directions.

2 Overview of TAAROA

TAAROA is a software middleware which tries to enable Grid systems (i.e., distributed systems using the Grid computing paradigm) to execute services and, at the same time, to preserve their SLA constraints. In this section we provide an high-level overview of the architecture of the TAAROA middleware.

As shown in Fig. 1, the architecture of TAAROA consists of five types of components: the TAAROA Client, the Information Service, the Repository Manager, the Scheduler and the Machine Manager (along with the associated Physical Machine). Each component is loosely coupled to each other, meaning that components, both of the same and of different type, weakly depend from each other.

The TAAROA Client component is the user interface to the TAAROA middleware; it is the component where all user requests start. For instance, it allows a user to execute a new service or to stop the execution of a running one.

The Information Service component is responsible for collecting, managing and publishing information about the state of the user services and of the other TAAROA components. This information is further used by TAAROA middleware in order to locate a particular TAAROA component or to get insights about its last published state.

The Machine Manager component is in charge of executing a user service on a Physical Machine. It uses the virtualization technology of the underlying Physical Machine in order to execute the service in an isolated environment and to avoid runtime dependencies issues: every service, along with its runtime environment, is encapsulated inside a Virtual Machine.

The Repository Manager component is responsible for hosting the images of the Virtual Machines associated to the user services.

The Scheduler component has the role of deciding to what Physical Machine a user service is to be assigned for the execution. In the current implementation of the TAAROA middleware, requests for service submission are queued and served in the order of their arrival, by assigning to each service the first available Physical Machine.

There are several type of interactions occurring between TAAROA components. In the following, just for illustration, we outline the workflow associated to the submission of a user service. When a user wants to execute a service to TAAROA, he accesses to the TAAROA Client, selects the wanted service from a list retrieved by the TAAROA Client from the Information Service, and finally submit it to TAAROA. In response to the service submission request, the TAAROA Client contacts the Scheduler for submitting the interested service. In turn, the Scheduler queries the Information Service for obtaining the list of the available Physical Machines, along with their allocation statistics, and chooses the best one where executing the user service. This choice can be done according to different strategies; the scheduling strategy actually implemented in TAAROA follows the FCFS (First-Come First-Served) policy both for selecting what service to execute and for choosing the Physical Machine where running it. After having chosen the Physical Machine, the Scheduler instructs the Repository Manager to execute the service on the chosen Physical Machine. In turn, the Repository Manager contacts the Ma-

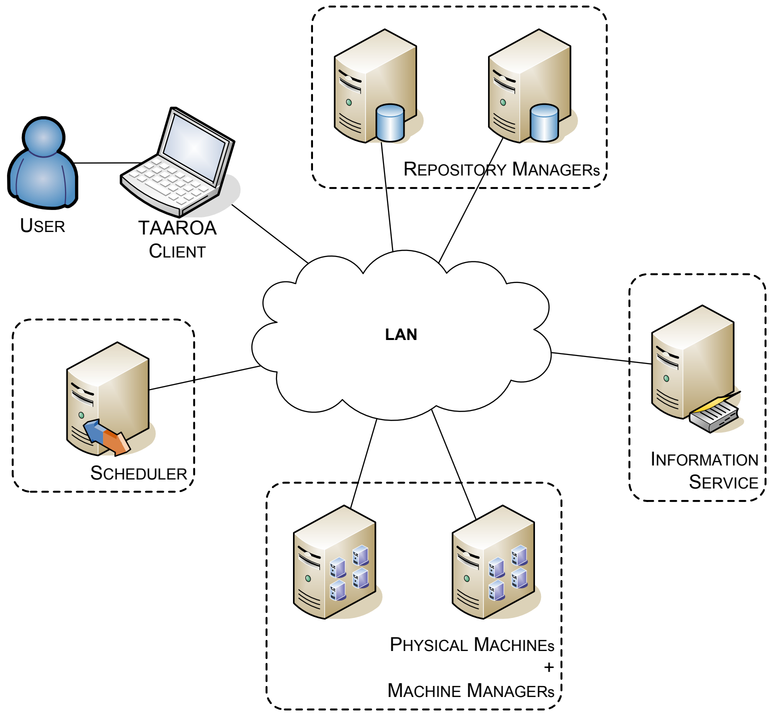


Figure 1. The component-level architecture of TAAROA.

chine Manager of the chosen Physical Machine and sends it the files related to the Virtual Machine representing the given service. The Machine Manager, upon completion of the Virtual Machine files transfer, instructs the virtualization layer of the Physical Machine where it is running to start the new Virtual Machine.

In the above workflow, whenever a TAAROA component needs to locate another one, the Information Service is contacted in order to obtain the needed information. The interactions occurring between the various components is describe in the following section, while the details about all of the exchanged information is delayed to §5.

3 Architecture

In this section we provide a detailed view of the architecture of TAAROA middleware. We begin by a thorough description of the concepts underlying the TAAROA middleware; then we describe the role of each TAAROA components; finally we provide an example of the most important interactions between TAAROA components.

3.1 Concepts

In this section we outline the fundamental concepts of the TAAROA architecture upon which all of the TAAROA components are based on. They include the concept of Physical Machine, Service and Virtual Machine.

3.1.1 Physical Machine

The *Physical Machine* concept represents a computer connected to a network and equipped with hardware and operating system that supports virtualization technologies.

The amount of its hardware components (e.g., RAM size, disk space, CPU clock frequency, network bandwidth, and so on) is sent by the Machine Manager to the Information Service when it enters into TAAROA. This information might be used by the Scheduler for understanding which Physical Machine best fits the performance needs of a Virtual Machine.

3.1.2 Service

The *Service* concept represents an application that usually waits for incoming client requests and replies with useful information. Typical example of Services are web servers, database servers, name servers and authentication servers. The execution of a Service is requested through a TAAROA Client and is performed on a Physical Machine inside a Virtual Machine. Along with the Service application, a Service usually comes with one or more predefined constraints called Service Level Agreement (SLA). An SLA is a formal

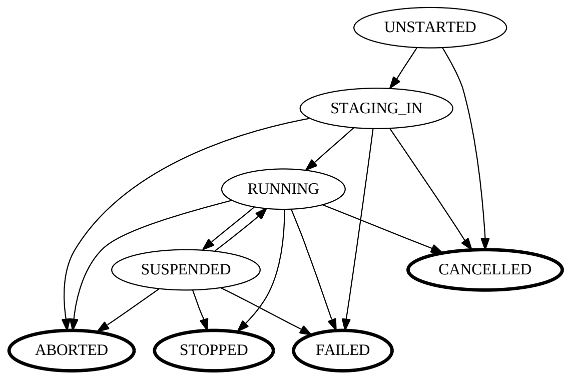


Figure 2. Execution state for a Virtual Machine.

description of the level of a Service, upon which two negotiating parties (the provider and the recipient) agree. It is commonly described in terms of Service Level Objectives (SLOs) and Service Level Specifications (SLSs), which in turn define temporal and performance metrics for measuring the level of Service, and the operational guidelines for achieving the desired level of Service, respectively. To illustrate, an SLA may specify the levels of availability, reliability and performance of the service and possible penalties in case of violation.

In TAAROA, the application runtime environment of a Service is generally stored as one or more files; for instance, it can represent an image (e.g., an ISO-9660 file) of an operating system. For each Service, its associated SLA is defined as a set of requirements on physical resources that needs to be satisfied once the related Service is chosen to be executed; for instance, a requirement can specify the minimum amount of space on a physical disk.

3.1.3 Virtual Machine

The *Virtual Machine* concept represents a Service submitted for execution. It is characterized by the following attributes: the instantiated Service defining a self-contained application environment, the address of the Physical Machine where the service is running, and the name or the IP address of the virtual host through which it is possible to communicate with the Virtual Machine.

The execution of a Virtual Machine can evolve through several states. In Fig. 2 is shown how execution states of a Virtual Machine are related to each other. Before running, the execution state of a Virtual Machine is UNSTARTED. Once the Virtual Machine is selected for execution, it changes its state to STAGING_IN; in this state, the Virtual Machine is sent to the Physical Machine where it will be executed. After the stage-in phase has been completed, the Virtual Machine is started and its execution state becomes RUNNING. Eventually, when the Virtual Machine is no more needed, the TAAROA Client can decide to shut it down, making the execution state of that Virtual Machine to switch to STOPPED. There are other possible execution states a Virtual Machine can take. Specifically, if the execution of a Virtual Machine is explicitly cancelled by a TAAROA Client, the corresponding execution state changes to CANCELLED. If the execution of a Virtual Machine is prematurely arrested by a TAAROA component (e.g., for the lack of physical resources needed by the Virtual Machine), the corresponding execution state changes to ABORTED. If something goes wrong during the execution of a Virtual Machine, the corresponding execution state changes to FAILED. The execution of a Virtual Machine can even be temporarily suspended; in this case the execution state changes to SUSPENDED. Once its execution is resumed, the execution state turns back to RUNNING.

Execution states divide in two main groups: *final* and *temporary* states. Final states are the ones that once a Virtual Machine enters, cannot leave any more. They include the ABORTED, CANCELLED, FAILED and STOPPED states (i.e., the ones marked with a thicker line in the figure). Instead, temporary states are the ones that a Virtual Machine temporarily takes before entering a final state. They include the RUNNING, STAGING_IN, SUSPENDED and UNSTARTED states. According to Fig. 2, the execution of a Virtual Machine, generally, cycles through one of more temporary states before entering a final state.

3.2 Components

In this section we provide a detailed description of the components of the TAAROA architecture. As described in §2 and shown on Fig. 1, there are five types of components: the TAAROA Client, the Information Service, the Repository Manager, the Scheduler and the Machine Manager (along with the associated Physical Machine). Each component is loosely coupled to each other, meaning that components, both of the same and of different type, weakly depend from each other. Components, in opposite to concepts, interact directly with the TAAROA system to achieve their goals. There are *active* and *passive* components. Active components are the ones that initiate interactions with the middleware; in TAAROA, the TAAROA Client, the Repository Manager and the Machine Manager are active components. Passive components are the ones that act as a consequence of stimulus sent by active components to the system; in TAAROA, the Information Service and the Scheduler are passive components.

3.2.1 Information Service

The *Information Service* component is responsible for collecting, managing and publishing information about the primary TAAROA entities: Physical Machine, Repository Manager, Service and Virtual Machine. The structure of the information stored in the Information Service follows the database schema described in §4.

Every TAAROA component can communicate with the Information Service. The Repository Manager component contacts the Information Service for registering or unregistering itself; in addition, it is responsible for inserting, updating and removing information about Services. The Machine Manager component interacts with the Information Service for registering or unregistering itself, the Physical Machine where it runs and the Virtual Machines hosted by the Physical Machine. The TAAROA Client component asks the Information Service to provide it the list of Services for choosing what of them to submit for execution. The Scheduler component queries the Information Service for obtaining the resource utilization of the available Physical Machines; this information can help it to decide on what machine a given Service is to be executed.

Actually, there is only one Information Service component in TAAROA. This means that information about the state of TAAROA resources are definitively managed in a centralized way.

3.2.2 Machine Manager

The *Machine Manager* component is a software component that runs on a Physical Machine and waits for connections.

The Machine Manager is in charge of managing each Virtual Machine that is hosted by the Physical Machine where it is running, by means of a software component named Virtual Machine Monitor (VMM). The VMM, also referred to as hypervisor, is a software layer that provides virtualization support and can run either directly on hardware, if that supports it, or on top of the operating system.

At startup the Machine Manager contacts the Information Service and sends it the hardware characteristics of the Physical Machine on which resides. When the Repository Manager is asked to start a Service, it sends to the Machine Manager the Virtual Machine image and the related configuration file. The Virtual Machine image contains all the files that build up the Service runtime environment, while the configuration file contains all the settings used by the VMM when starting, stopping and suspending the Virtual Machine.

A Machine Manager is also responsible for keeping the Information Service up to date by communicating changes in the amount of resources allocated to every managed Virtual Machine.

3.2.3 Repository Manager

The *Repository Manager* component is a software component that is responsible for hosting Virtual Machine images. It has access to a storage area (either local or remote) where each Virtual Machine image is placed, along with its configuration file ad everything that is necessary for running the Virtual Machine.

Upon service submission request, it takes care of sending the corresponding Virtual Machine image and related files, to a given Physical Machine and asks the Machine Manager (located on that machine) to start the Virtual Machine. In a similar way, when it is asked to stop a Virtual Machine, it instructs the Machine Manager (where the Virtual Machine is running) accordingly.

A Repository Manager is also responsible for keeping the Information Service up to date by communicating changes in the execution state of every managed Virtual Machine.

3.2.4 Scheduler

The *Scheduler* component is responsible for deciding to what Physical Machine a Service is to be assigned for the execution. This decision is taken through the so called scheduling policy, often referred to as scheduling heuristic. In the current implementation of the TAAROA middleware, the only available scheduling heuristic is the one that uses the First-Come-First-Served (FCFS) policy: requests for Service submission are queued and served in the order of their arrival, by assigning to each Service the first available Physical Machine.

Actually, there is only one Scheduler component in TAAROA. This means that, from the point of view of a TAAROA Client, the Scheduler component is the only single point of control for starting, monitoring and stopping the execution of a Service in TAAROA: every request for submitting, managing and terminating a Service must go through the Scheduler component.

3.2.5 TAAROA Client

The *TAAROA Client* component communicates with the other TAAROA components in order to submit Services and to monitor their execution. It represents the user interface to the TAAROA middleware: all of the interactions occurring between a user (or a user application) and TAAROA happen through this component.

The *TAAROA Web Portal* is a specific type of TAAROA Client which offers an high-level user-friendly interface in order to make the interactions with the other TAAROA components easier from the point of view of its users.

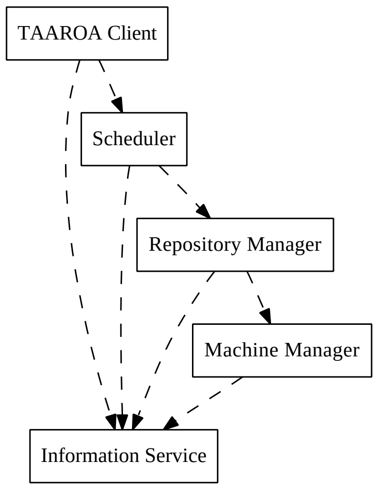


Figure 3. Static relations between TAAROA components.

3.3 Workflow

In this section we describe the static relations and the dynamic interactions between the TAAROA components.

Fig. 3 shows the structural relations between TAAROA components. The TAAROA Client uses the Information Service for getting access to information about Services and other related entities; for instance, it queries the Information Service for retrieving the list of available Services. In addition, the TAAROA Client uses the Scheduler for executing, managing and stopping one or more Services. The Scheduler uses the Information Service for obtaining information about Services and resource utilization of Physical Machines. This information might be used, for example, for deciding to what Physical Machine a given Service is to be assigned for the execution. Moreover, the Scheduler uses the Repository Manager for submitting and stopping a specific Service. The Repository Manager uses the Information Service for registering and unregistering itself along with the published Services. The Repository Manager also uses the Machine Manager for controlling the execution of a Virtual Machine. Finally, the Machine Manager uses the Information Service for registering and unregistering the Physical Machine on which it runs and every Virtual Machine it manages.

The dynamic interactions between TAAROA components are based on a message-oriented and stateless protocol; this means that each pair of TAAROA components exchanges messages with the rest of TAAROA components, in the form of request-reply messages, and each message neither depends on previously sent messages nor on additional information stored on the receiving component.

In the rest of this section we present the workflow concerning two of the most important TAAROA interactions: the Service submission workflow and the Service stopping workflow. The type of modelling diagram used for describing these interactions is the UML *communication diagram* [6]. This kind of diagram models the interactions between TAAROA components by showing the flow of messages exchanged among them. In order to maintain the sequential ordering of interactions, messages are labeled with a chronological number (usually starting from one), placed near the link where the message is sent over.

In the Service submission workflow, schematically shown on Fig. 4, the active actor is the TAAROA Client. When it wants to submit a Service, the first action it performs is to request to the Information Service component the list of the available Services. After having chosen the Service it wants to run, denoted with Service x in the figure, it sends a request to the Scheduler component for submitting the selected Service and waits for a replay. In turn, the Scheduler contacts the Information Service for obtaining the list of Physical Machines along with their allocation statistics. The information needed to calculate these statistics is sent to the Information Service by the Machine Manager both when the Physical Machine, on which it is running, is added to TAAROA and when one of the Virtual Machine, running on that Physical Machine, changes its execution state. These statistics can be used by the Scheduler for deciding, according to a proper scheduling heuristic, if a Service can be immediately executed and on what Physical Machine. If the Service can be executed and a suitable Physical Machine is found, the Scheduler instructs the Repository Manager to submit the Virtual Machine associated to this Service on the chosen Physical Machine. In the figure, the Physical Machine chosen by the Scheduler is named y. Then, the Repository Manager contacts the Machine Manager of the chosen Physical Machine for starting a new Virtual Machine for the given Service. It sends all the files composing the Virtual Machine to the Machine Manager which, in turn, instructs the virtualization layer to start the Virtual Machine. In the figure, the new Virtual Machine is marked as z. In case of success, it registers the newly created Virtual Machine to the Information Service along with its parameters (network configuration, administration credentials and so on), for later retrieval, and notifies the Repository Manager about the starting of the Virtual Machine. When the Repository Manager receives the notification about the execution of the Virtual Machine, it contacts the Information Service, for updating the execution status of that Virtual Machine, and the Scheduler, for passing to it the Virtual Machine global identifier (i.e., the value that uniquely identifies the Virtual Machine inside the TAAROA system) obtained from the Information Service. The Scheduler, in turn, notifies the requesting TAAROA Client, that

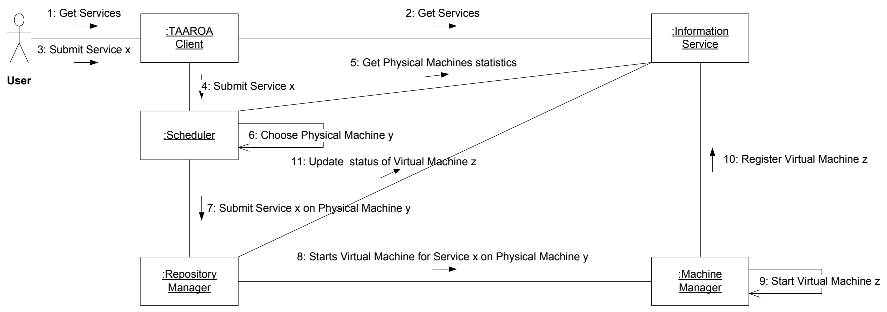

1: Get Services
3: Submit Service x
User
:TAAROA Client
2: Get Services
:Information Service
5: Get Physical Machines statistics
4: Submit Service x
:Scheduler
6: Choose Physical Machine y
11: Update status of Virtual Machine z
10: Register Virtual Machine z
7: Submit Service x on Physical Machine y
8: Starts Virtual Machine for Service x on Physical Machine y
:Repository Manager
:Machine Manager
9: Start Virtual Machine z

Figure 4. The communication diagram for TAAROA service submission.

is the one that initially began the workflow. In the above figure, all the replies are omitted for the sake of simplicity.

The other important interaction between TAAROA components is the stopping of a Service. In Fig. 5 is depicted the workflow for stopping a Service. Likewise the Service submission workflow, the active actor is the TAAROA Client. When it wants to stop a running Service, identified as z in the figure, it sends a request to the Scheduler component. Subsequently, the Scheduler contacts the Repository Manager component which, in turn, queries the Information Service for finding out what Physical Machine is hosting that Virtual Machine. Once the Repository Manager obtains the requested information, it asks the Machine Manager, running on that Physical Machine, to stop the involved Virtual Machine. In the figure, the hosting Physical Machine is denoted with y. The Machine Manager delegates the Virtual Machine Monitor to stop that Virtual Machine, then unregisters the Virtual Machine on the Information Service and, on success, notifies the Repository Manager about the stopping of the Virtual Machine. Finally, the Repository Manager updates the execution status of the stopped Virtual Machine and then notifies the Scheduler, which successively notifies the requesting TAAROA Client. Similarly to the diagram for the Service submission workflow, in the above figure, all the replies are omitted for the sake of simplicity.

The other kind of interactions that might occur between TAAROA components mainly concern the updating of information kept in the Information Service. For example, each Repository Manager registers or unregisters itself to the Information Service whenever it joins to or leaves the TAAROA system, respectively. Likewise, when a Machine Manager joins to or leaves the TAAROA system, it registers or unregisters itself, respectively, to the Information Service component, along with the Physical Machine on which it runs. Moreover, every time a new Service is added or an existing Service is removed from TAAROA, it is stored on or deleted from a Repository Manager which, in turn, takes care of registering or unregistering it to the Information Service, respectively. When a Virtual Machine changes the state of its execution, the Repository Manager updates the related information maintained by the Information Service accordingly.

4 Database

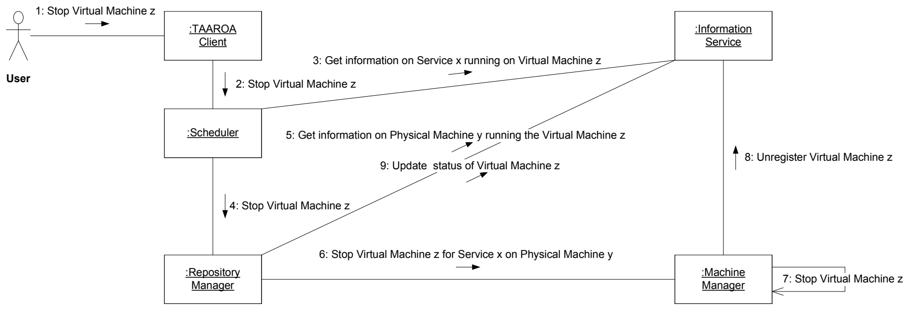
In this section we describe the databases maintained by the Information Service, the Repository Manager and the Machine Manager components. The modelling diagram used for showing the different database models is the Data Structure Diagram (DSD). This kind of diagram is an extension of the classic Entity-Relationship (E-R) diagram [3]; it differs from it in that the E-R model focuses on the relationships between different entities, whereas a DSD focuses on the relationships of the elements within an entity, enabling users to better understand the links and the relationships between each entity. In this diagram, entities are represented as boxes, entity attributes are specified inside the entity boxes, while binary relationships are drawn as lines connecting the boxes representing the participating entities. For n-ary relationships, an additional entity is used; it might have attributes which specify the constraints that bind participating entities together. The cardinality of an entity for a particular relationship is expressed using the “crow’s foot” notation.

4.1 The *Information Service* database

In Fig. 6 is shown the DSD of the database used by the Information Service component. In the rest of this section, we describe the entities and the relationships contained in this database.

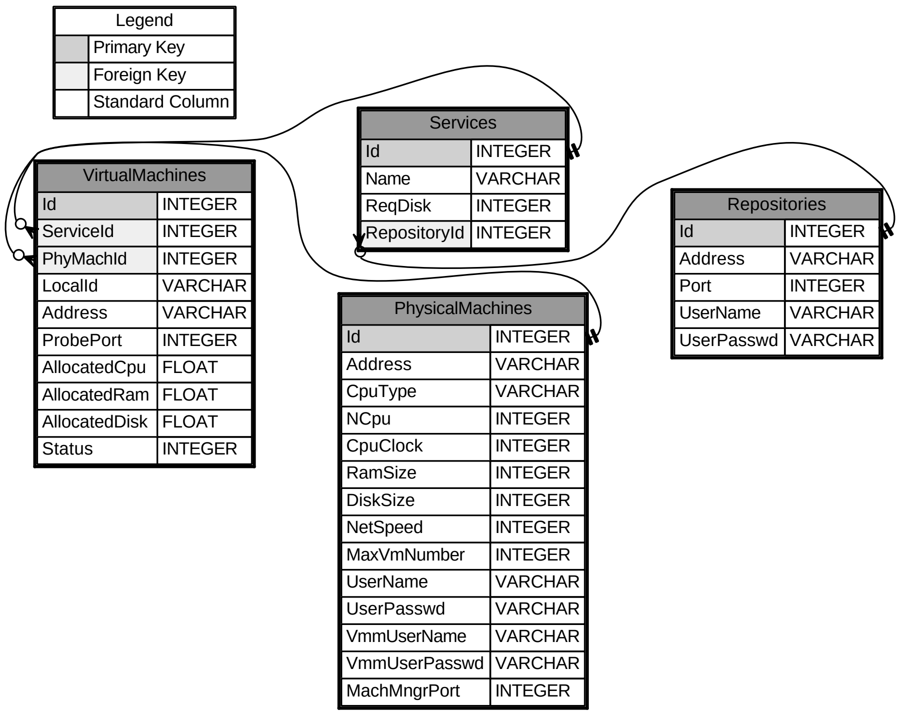
4.1.1 The *PhysicalMachines* entity

The *PhysicalMachines* entity represents the Physical Machine concept presented in §3.1.1. A PhysicalMachines entity is uniquely identified by an integer positive number represented by the attribute *Id*. A Physical Machine can host one or more Virtual Machines, whereas a Virtual Machine is



1: Stop Virtual Machine z
User
:TAAROA Client
:Information Service
3: Get information on Service x running on Virtual Machine z
2: Stop Virtual Machine z
:Scheduler
5: Get information on Physical Machine y running the Virtual Machine z
9: Update status of Virtual Machine z
8: Unregister Virtual Machine z
4: Stop Virtual Machine z
6: Stop Virtual Machine z for Service x on Physical Machine y
:Repository Manager
:Machine Manager
7: Stop Virtual Machine z

Figure 5. The communication diagram for TAAROA service stopping.



Legend	
▨	Primary Key
░	Foreign Key
□	Standard Column

Services	
Id	INTEGER
Name	VARCHAR
ReqDisk	INTEGER
RepositoryId	INTEGER

VirtualMachines	
Id	INTEGER
ServiceId	INTEGER
PhyMachId	INTEGER
LocalId	VARCHAR
Address	VARCHAR
ProbePort	INTEGER
AllocatedCpu	FLOAT
AllocatedRam	FLOAT
AllocatedDisk	FLOAT
Status	INTEGER

Repositories	
Id	INTEGER
Address	VARCHAR
Port	INTEGER
UserName	VARCHAR
UserPasswd	VARCHAR

PhysicalMachines	
Id	INTEGER
Address	VARCHAR
CpuType	VARCHAR
NCpu	INTEGER
CpuClock	INTEGER
RamSize	INTEGER
DiskSize	INTEGER
NetSpeed	INTEGER
MaxVmNumber	INTEGER
UserName	VARCHAR
UserPasswd	VARCHAR
VmmUserName	VARCHAR
VmmUserPasswd	VARCHAR
MachMngrPort	INTEGER

Figure 6. The E-R schema of the Information Service database.

running on only one Physical Machine at a time. The maximum amount of Virtual Machines a given Physical Machine can run is specified by the attribute *MaxVmNumber*.

The other attributes concern hardware and system properties and administration information. Regarding the hardware and system characteristics, the attribute *CpuType* represents the vendor and the model of the CPU, the attribute *NCpu* denotes the number of cores or processors installed on the Physical Machine, while the attribute *CpuClock* specifies the CPU clock frequency (in MegaHertz). The attributes *RamSize*, *DiskSize* and *NetSpeed* represent respectively the amount of system RAM (in MegaBytes), the amount of disk space (in MegaBytes) and the speed of the network card (in MegaBps) of the Physical Machine. Finally, the attribute *Address* represents the IP address of the Physical Machine.

For what concerns the information for administration purpose, the attributes *UserName* and *UserPassword* represent the credentials for remotely accessing to the Physical Machine, the attribute *VmmUserName* and *VmmUserPassword* are the credentials for gaining access to the Virtual Machine Monitor and the attribute *MachMngrPort* is the port at which the Machine Manager waits for requests.

4.1.2 The *Repositories* entity

The *Repositories* entity represents the Repository Manager concept stated in §3.2.3. A Repositories entity is uniquely identified by the attribute *Id*, which is an integer positive number. The attributes *Address* and *Port* are used for connecting to the Repository Manager, while attributes *UserName* and *UserPasswd* are the credentials needed for gaining access to it.

4.1.3 The *Services* entity

The *Services* entity models the Service concept described in §3.1.2. A Services entity is uniquely identified by an integer positive number represented by the attribute *Id*. This entity is uniquely associated to a Repositories entity (through the attribute *RepositoryId*), meaning that a Service is provided by one and only one Repository Manager component. Furthermore, a Services entity can participate in the association with one more VirtualMachines entities, but a VirtualMachines entity is associated to exactly one Services entity. This basically means that the same Service can appear in one or more Virtual Machines, but a Virtual Machine can only run exactly one Service. The remaining attributes are the Service name (attribute *Name*) and the amount of disk space (in bytes) needed by the Service for executing (attribute *ReqDisk*).

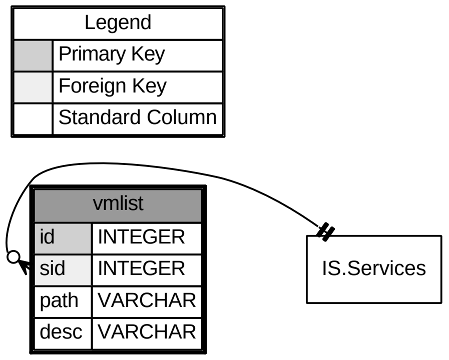


Figure 7. The E-R schema of the Repository Manager database.

4.1.4 The *VirtualMachines* entity

The *VirtualMachines* entity describes the Virtual Machine concept outlined in §3.1.3. A VirtualMachines entity is uniquely identified by the attribute *Id*, which is an integer positive number. A Virtual Machine represents a solely running Service instance and can live in only one Physical Machine (though Virtual Machine migration can change the hosting machine along the time); the attributes that link a Virtual Machine to its Service and to its Physical Machine are *ServiceId* and *PhyMachId*, respectively. On the other hand, more than one Virtual Machine can execute the same Service and a Physical Machine may contain several Virtual Machines. Among the other attributes characterizing this entity, those that are worth noting are the ones indicating the amount of physical resources allocated to a Virtual Machine and the state of the execution. The resource allocation attributes include the CPU share allocation (attribute *AllocatedCpu*), the fraction of allocated system memory (attribute *AllocatedRam*) and the fraction of allocated disk space (attribute *AllocatedDisk*). For what concerns the execution state of a Virtual Machine, it is represented by the attribute *Status*, an integer number whose possible values are defined according to the TAAROA communication protocol (described in §5).

4.2 The *Repository Manager* database

In Fig. 7 is shown the DSD of the database used by the Repository Manager component. The purpose of this database is to store information that allow to associate TAAROA global descriptors with information that are local to each Repository Manager. For this reason, each Repository Manager maintains a different copy of this database.

In the rest of this section, we describe the entities and the relationships contained in this database.

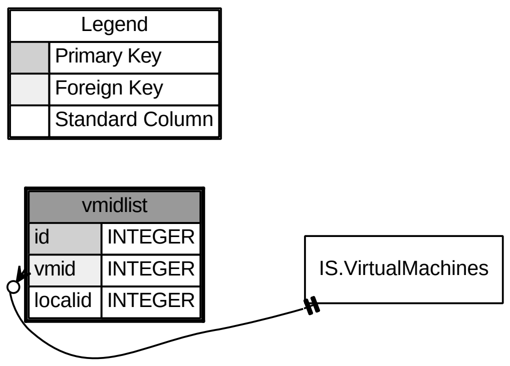


Figure 8. The E-R schema of the Machine Manager database.

Symbol	Description
<#>	The literal character '#'.
<b>	A sequence of one or more blank characters (whitespace or horizontal tabulation).
$D_{64}[s]$	The base64 decoding of the string s.
$E_{64}[s]$	The base64 encoding of the string s.
IS	Abbreviation for Information Server.
MM	Abbreviation for Machine Manager.
RM	Abbreviation for Repository Manager.
SC	Abbreviation for Scheduler.
SVC	Abbreviation for Service.
TC	Abbreviation for TAAROA client.
VM	Abbreviation for Virtual Machine.
WP	Abbreviation for TAAROA Web Portal.

Table 1. Notations and abbreviations for the TAAROA communication protocol.

4.2.1 The *vmlist* entity

The *vmlist* entity is used by the Repository Manager for retrieving, from a given Service identifier, all of the files composing a Virtual Machine. The attribute *sid* represents the Service identifier related to a particular Virtual Machine; it is a foreign key referring to the attribute *Id* of the entity *Services*, stored in the Information Service database (see §4.1.3). The *path* attribute is the actual path where all the files for a given Virtual Machine are stored.

4.3 The *Machine Manager* database

In Fig. 8 is shown the DSD of the database used by the Machine Manager component. This database contains only information that is local to each Machine Manager; for instance, the information regarding Virtual Machines is restricted only to the ones running on the Physical Machine on which the Machine Manager resides. For this reason, each Machine Manager maintains a different copy of this database.

In the rest of this section, we describe the entities and the relationships contained in this database.

4.3.1 The *vmidlist* entity

The *vmidlist* entity is used by the Machine Manager for associating a Virtual Machine concept (see §3.1.3) with a real Virtual Machine implementation. Specifically, it links a Virtual Machine global identifier, assigned by the Information Service, to a local identifier associated to the corresponding Virtual Machine running on the Physical Machine on which the Machine Manager resides; this local identifier is assigned to the real Virtual Machine by the underlying Virtual Machine Monitor. Each vmidlist entity is uniquely identified by an integer positive number represented by the attribute *Id*. The attribute *vmid* represents the TAAROA Virtual Machine identifier, whereas the attribute *localid* is the local Virtual Machine identifier assigned by the Virtual Machine Monitor.

5 Communication Protocol

In this section we describe the communication protocol used by the current version of TAAROA middleware. The protocol is at the base of all the dynamic interactions occurring between TAAROA components; it is a message-oriented and stateless protocol, that is each TAAROA component exchanges with the others a series of request-reply messages that neither depend on previously sent messages nor on additional information stored on the receiving component.

The specification of the protocol messages follows precise symbol and number conventions. In Tab. 1 are shown the symbols, along with their meaning, employed for describing the format of the protocol messages. Instead, in §5.1, the format of the number, the unit of measurement and the other constants is illustrated.

5.1 Common formats

5.1.1 Integer Numbers representation

The protocol supports the following integer number format (expressed as POSIX regular expression):

- $\backslash d+$ (e.g. 14).

No negative value is allowed.

5.1.2 Real Numbers representation

The protocol supports the following real number formats (expressed as POSIX regular expression):

- Standard notation: $\backslash d + \backslash . \backslash d+$ (e.g. 14.5).
- Scientific notation: $\backslash d + \backslash . \backslash d + [eE][+-]\backslash d+$ (e.g. $1.45e + 1$).

No negative value is allowed.

5.1.3 Frequency Unit of Measurement representation

The string MUST be an integer number optionally followed by a unit specifier character. Possible unit specifiers are:

- `Hz` for Hertz.
- `KHz` for KiloHertz.
- `MHz` for MegaHertz.
- `GHz` for GigaHertz.
- `THz` for TeraHertz.
- `PHz` for PetaHertz.

If no unit specifier character is specified, the default value depends on the context where the unit of measurement has to be specified.

5.1.4 Memory Unit of Measurement representation

The string MUST be an integer number optionally followed by a unit specifier character. Possible unit specifiers are:

- `B` for bytes.
- `KB` for Kilobytes.
- `MB` for Megabytes.
- `GB` for Gigabytes.
- `TB` for Terabytes.
- `PB` for Petabytes.

If no unit specifier character is specified, the default value depends on the context where the unit of measurement has to be specified.

5.1.5 Net Speed Unit of Measurement representation

The string MUST be an integer number §5.1.1 optionally followed by a unit specifier character. Possible unit specifiers are:

- `bps` for bits-per-second (bit/s).
- `Kbps` for Kilobps (Kbit/s).
- `Mbps` for Megabps (Mbit/s).
- `Gbps` for Gigabps (Gbit/s).
- `Tbps` for Terabps (Tbit/s).
- `Pbps` for Petabps (Pbit/s).

If no unit specifier character is specified, the default value depends on the context where the unit of measurement has to be specified.

5.1.6 Execution Status Codification

The execution status of a Virtual Machine is coded as an integer number:

`0`: represents the UNKNOWN execution status.

`1`: represents the UNSTARTED execution status.

`2`: represents the READY execution status.

`3`: represents the STAGING_IN execution status.

`4`: represents the RUNNING execution status.

`5`: represents the SUSPENDED execution status.

`6`: represents the STOPPED execution status.

`7`: represents the CANCELLED execution status.

`8`: represents the FAILED execution status.

`9`: represents the ABORTED execution status.

5.2 Messages issued to the Information Server

5.2.1 GETPHYMACH – *Physical Machine details* request

Sent by a RM to the IS for getting information about a specific physical machine.

⟨ **`GETPHYMACH`**`<b>PHY_ID` ⟩

where:

- `PHY_ID`: integer number §5.1.1 representing the physical machine identifier.

Possible replies from the IS are:

- In case of success:

  ⟨ **OK**<b>PHY_IP<b>MM_PORT ⟩

  where:

  – PHY_IP: string containing the IP address of the requested physical machine.
  – MM_PORT: integer number §5.1.1 representing the TCP port of the MM.

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.2 GETVM – *Virtual Machine details* request

Sent by a TC to the IS when it wants to know the details regarding a given submitted service (virtual machine).

⟨ **GETVM**<b>VM_ID ⟩

where:

- VM_ID: integer number §5.1.1 containing the submitted service (virtual machine) identifier.

Possible replies from the IS are:

- In case of success:

  ⟨ **OK**<b>S_ID<b>PHY_ID<b>
  VM_LOCAL_ID<b>VIRT_IP<b>STATUS ⟩

  where:

  – S_ID: integer number §5.1.1 containing the service identifier.
  – PHY_ID: integer number §5.1.1 containing the identifier of the physical machine.
  – VM_LOCAL_ID: string containing the identifier used by the MM to uniquely retrieve a VM.
  – VIRT_IP: string containing the IP address of the virtual machine on which the service is running.
  – STATUS: integer number §5.1.1 representing the execution status of the submitted service (§3.1.3).

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.3 GETVMMACHMNGR – *Virtual Machine Machine Manager* request

Sent by a RM (or other clients) to the IS when it wants to know the machine manager associated to a given submitted service (virtual machine).

⟨ **GETVMMACHMNGR**<b>VM_ID ⟩

where:

- VM_ID: integer number §5.1.1 containing the submitted service (virtual machine) identifier.

Possible replies from the IS are:

- In case of success:

  ⟨ **OK**<b>PHY_ID<b>PHY_IP<b>
  MM_PORT<b>VM_LOCAL_ID ⟩

  where:

  – PHY_ID: integer number §5.1.1 representing the identifier of the Physical Machine where the MM is running.
  – PHY_IP: string representing the IP address of the Physical Machine where the MM is running.
  – MM_PORT: integer number §5.1.1 representing the TCP port of the MM.
  – VM_LOCAL_ID: string containing the identifier used by the MM to uniquely retrieve a VM.

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.4 GETVMSERV – *Virtual Machine Service* request

Sent by a TC to the IS when it wants to know the service associated to a given submitted service (virtual machine).

⟨ **GETVMSERV**<b>VM_ID ⟩

where:

- VM_ID: integer number §5.1.1 containing the submitted service (virtual machine) identifier.

Possible replies from the IS are:

- In case of success:

  ⟨ **OK**<b>S_ID<b>E_{64}[NAME]<b>
  RM_ID<b>RM_IP<b>RM_PORT ⟩

  where:

  – S_ID: integer number §5.1.1 representing the service identifier.

  - NAME: string representing the symbolic name of the service.
  - RM_ID: integer number §5.1.1 representing the RM identifier.
  - RM_IP: string representing the IP address of the RM.
  - RM_PORT: integer number §5.1.1 representing the TCP port of the RM.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.5 GETVMSTATUS – *Virtual Machine Status* request

Sent by a TC to the IS when it wants to know the execution status of a given submitted service (virtual machine).

⟨ **GETVMSTATUS**<b>VM_ID ⟩

where:

- VM_ID: integer number §5.1.1 containing the submitted service (virtual machine) identifier.

Possible replies from the IS are:

- In case of success:

  ⟨ **OK**<b>STATUS ⟩

  where STATUS is an integer number §5.1.1 representing the service execution status (§3.1.3).
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.6 LISTPHYMACH – *List of Physical Machines* request

Sent by a TC to the IS when it wants to retrieve the list of all registered physical machines.

⟨ **LISTPHYMACH** ⟩

Possible replies from the IS are:

- In case of success, returns:

  ⟨ **OK**<b>PhyMachList ⟩

  where PhyMachList is a list of entry messages:

  ⟨ PHY_ID<b>PHY_IP<b>MM_PORT ⟩

  where:

  - PHY_IP: string containing the IP address of a physical machine.
  - PHY_ID: integer number §5.1.1 representing the identifier of a physical machine.
  - MM_PORT: integer number §5.1.1 representing the TCP port of the MM.

  terminated by a dot message:

  ⟨ . ⟩

  indicating the end of the list. In case of empty list the following message is returned:

  ⟨ **OK**<b>. ⟩
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.7 LISTPHYMACHSTATUS – *List of Physical Machines along with Resource Utilization* request

Sent by a SC (or other clients) to the IS when it wants to retrieve the list of all registered physical machines along with the status of their resources utilization.

⟨ **LISTPHYMACHSTATUS** ⟩

Possible replies from the IS are:

- In case of success, returns:

  ⟨ **OK**<b>PhyMachList ⟩

  where PhyMachList is a list of entry messages:

  ⟨ PHY_ID<b>AVAIL_CPU<b>AVAIL_RAM<b>
  AVAIL_DISK<b>NETSPEED ⟩

  terminated by a dot message:

  ⟨ . ⟩

  indicating the end of the list.

  The fields in each entry has the following meaning:

  - PHY_ID: integer number §5.1.1 representing the identifier of a physical machine.
  - AVAIL_CPU: real number §5.1.2 representing the available number of processors expressed as a fraction of the total number of CPU/Core processors:

    $$NumOfCpus(PHY\_ID) - \sum_{\substack{VM\_ID \\ \text{on } PHY\_ID}} AllocCpuFrac(VM\_ID)$$

    Admissibile values are in the range of $[0, NumOfCpus(PHY\_ID)]$.

- AVAIL_RAM: real number §5.1.2 representing the available RAM expressed as a fraction of the total RAM size:

$$1 \quad - \quad \sum_{\substack{VM\_ID \\ \text{on } PHY\_ID}} AllocRamFrac(VM\_ID)$$

Admissibile values are in the range of $[0, 1]$.

- AVAIL_DISK: real number §5.1.2 representing the available disk expressed as a fraction of the total disk size:

$$1 \quad - \quad \sum_{\substack{VM\_ID \\ \text{on } PHY\_ID}} AllocDiskFrac(VM\_ID)$$

Admissibile values are in the range of $[0, 1]$.

- NETSPEED: string representing the total speed of the network interface card, expressed as an integer number followed by a unit of measurement specifier §5.1.5.

In case of empty list the following message is returned:

⟨ **OK**<b>. ⟩

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.8 LISTREPO – *List of Repositories* request

Sent by a TAAROA component to the IS when it wants to know the list of available RMs.

⟨ **LISTREPO** ⟩

Possible replies from the IS are:

- In case of success, returns:

  ⟨ **OK**<b>RepoList ⟩

  where RepoList is a list of entry messages:

  ⟨ REPO_ID<b>IP_ADDR<b>PORT<b> E_{64} [USER_NAME]<b>E_{64} [PASSWD] ⟩

  where:

  - IP_ADDR: string containing the IP address of the repository manager service.
  - PORT: integer number §5.1.1 representing the TCP port on which the repository manager service must be contacted.
  - USER_NAME: string containing the username that must be used to authenticate with the RM.
  - PASSWD: string containing the password that must be used to authenticate with the RM.

  terminated by a dot message:

  ⟨ . ⟩

  indicating the end of the list. In case of empty list the following message is returned:

  ⟨ **OK**<b>. ⟩

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.9 LISTSERV – *List of Services* request

Sent by a TC to the IS when it wants to retrieve the list of all registered services.

⟨ **LISTSERV** ⟩

Possible replies from the IS are:

- In case of success, returns:

  ⟨ **OK**<b>ServList ⟩

  where ServList is a list of entry messages:

  ⟨ S_ID<b>E_{64} [NAME]<b>RM_ID<b> RM_IP<b>RM_PORT ⟩

  where:

  - S_ID: integer number §5.1.1 representing the service identifier.
  - NAME: string representing the symbolic name of the service.
  - RM_ID: integer number §5.1.1 representing the RM identifier.
  - RM_IP: string representing the IP address of the RM.
  - RM_PORT: integer number §5.1.1 representing the TCP port of the RM.

  terminated by a dot message:

  ⟨ . ⟩

  indicating the end of the list. In case of empty list the following message is returned:

  ⟨ **OK**<b>. ⟩

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.10 LISTVM – *List of Virtual Machines from Service* request

Sent by a TC to the IS when it wants to retrieve the list of all submitted services (virtual machines) for the given service `S_ID`.

$\langle$ **LISTVM**`<b>S_ID` $\rangle$

where:

- `S_ID`: integer number §5.1.1 representing the service identifier.

Possible replies from the IS are:

- In case of success, returns:

  $\langle$ **OK**`<b>VMList` $\rangle$

  where `VMList` is a list of entry messages:

  $\langle$ `VM_ID<b>PHY_ID<b>VM_LOCAL_ID<b>VIRT_IP<b>STATUS` $\rangle$

  where:

  – `VM_ID`: integer number §5.1.1 containing the submitted service (virtual machine) identifier.
  – `PHY_ID`: integer number §5.1.1 containing the identifier of the physical machine.
  – `VM_LOCAL_ID`: string containing the identifier used by the MM to uniquely retrieve a VM.
  – `VIRT_IP`: string containing the IP address of the virtual machine on which the service is running.
  – `STATUS`: integer number §5.1.1 representing the execution status of the submitted service (§3.1.3).

  terminated by a dot message:

  $\langle$. $\rangle$

  indicating the end of the list. In case of empty list the following message is returned:

  $\langle$ **OK**`<b>.` $\rangle$

- $\langle$ **ERR**`<b>CODE` $\rangle$ otherwise, where `CODE` is an integer number representing an error code.

5.2.11 REGPHYMACH – *Physical Machine Registration* request

Sent by a MM to the IS for registering a specific physical machine.

$\langle$ **REGPHYMACH**`<b>PHY_IP<b>`E_{64} [`CPUTYPE`]`<b>NCPU<b>CPUCLOCK<b>RAMSIZE<b>DISKSIZE<b>NETSPEED<b>MAX_VM_NUMBER<b>`E_{64} [`MACH_USERNAME`]`<b>`E_{64} [`MACH_PASSWORD`]`<b>`E_{64} [`XM_USERNAME`]`<b>`E_{64} [`XM_PASSWORD`]`<b>MM_PORT` $\rangle$

where:

- `PHY_IP`: string representing the physical machine IP address.
- `CPUTYPE`: string representing the model or architecture or type of the CPU installed on the machine.
- `NCPU`: integer number §5.1.1 representing the total number of CPU processors/cores installed on the machine.
- `CPUCLOCK`: string representing the clock frequency of a single CPU processor/core of the machine. See §5.1.3 for the specification of frequency unit of measurement. If no unit specifier character is specified, the *MegaHertz* unit of measurement is assumed as default.
- `RAMSIZE`: string representing the total memory available on the machine. See §5.1.4 for the specification of memory unit of measurement. If no unit specifier character is specified, the *Megabyte* unit is assumed as default.
- `DISKSIZE`: string representing the total disk space available on the machine. See §5.1.4 for the specification of memory unit of measurement. If no unit specifier character is specified, the *Megabyte* unit is assumed as default.
- `NETSPEED`: string representing the speed of the (main) network card installed on the machine. See §5.1.5 for the specification of net speed unit of measurement. If no unit specifier character is specified, the *Mbit/s* unit is assumed as default.
- `MAX_VM_NUMBER`: integer number §5.1.1 representing the maximum allowed number of running virtual machines. The value -1 means "no limit".
- `MACH_USERNAME`: string representing the name of the user used for logging in the machine.
- `MACH_PASSWORD`: string representing the password of the user used for logging in the machine.
- `XM_USERNAME`: string representing the name of the user used for controlling the Xen Manager.
- `XM_PASSWORD`: string representing the password of the user used for controlling the Xen Manager.

- MM_PORT: integer number §5.1.1 representing the TCP port number where the MM is accepting connections.

Possible replies from the IS are:

- ⟨ **OK**<b>PHY_ID ⟩ in case of success, where PHY_ID is the integer identifier of the new registered physical machine.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.12 REGREPO – *Repository Manager Registration* request

Sent by a RM to the IS for registering itself.

⟨ **REGREPO**<b>IP_ADDR<b>PORT<b> E_{64} [USER_NAME]<b>E_{64} [PASSWD] ⟩

where:

- IP_ADDR: string containing the IP address of the RM.
- PORT: integer number §5.1.1 containing the TCP port on which the RM waits for requests.
- USER_NAME: string containing the username that must be used to authenticate with the RM.
- PASSWD: string containing the password that must be used to authenticate with the RM.

Possible replies from the IS are:

- ⟨ **OK**<b>RM_ID ⟩ in case of success, where RM_ID is the integer identifier of the new registered RM.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.13 REGSERV – *Service Registration* request

Sent by a RM to the IS when it wants to register a new service.

⟨ **REGSERV**<b>RM_ID<b>E_{64} [NAME]<b>REQ_DISK ⟩

where:

- RM_ID: integer number §5.1.1 containing the RM identifier.
- NAME: string containing the symbolic name of the service.
- REQ_DISK: string representing the disk requirements. See §5.1.4 for the specification of disk unit of measurement. If no unit specifier character is specified, the *Kilobyte* unit is assumed as default.

Possible replies from the IS are:

- ⟨ **OK**<b>S_ID ⟩ in case of success, where S_ID is the integer identifier of the new registered service.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.14 REGVM – *Virtual Machine Registration* request

Sent by a MM to the IS when it wants to register a running VM (i.e., a VM that has been started on a physical machine).

⟨ **REGVM**<b>S_ID<b>PHY_ID<b>VM_LOCAL_ID<b> VIRT_IP<b>ALLOCATED_CPU<b> ALLOCATED_RAM<b>ALLOCATED_DISK ⟩

where:

- S_ID: integer number §5.1.1 containing the service identifier.
- PHY_ID: integer number §5.1.1 containing the identifier of the physical machine.
- VM_LOCAL_ID: string containing an identifier used by the MM to uniquely retrieve a VM.
- VIRT_IP: string containing the IP address of the physical machine on which the VM is running.
- ALLOCATED_CPU: real number §5.1.2 representing the number of CPU/Core processors allocated to the VM.
- ALLOCATED_RAM: real number §5.1.2 representing the amount of RAM allocated to the VM.
- ALLOCATED_DISK: real number §5.1.2 representing the amount of disk allocated to the VM.

Possible replies from the IS are:

- ⟨ **OK**<b>VM_ID ⟩ in case of success, where VM_ID is the integer identifier of the new registered virtual machine.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.15 SRVPROTOVER – *Protocol Version* request

Sent by a client to the IS for getting information about the TAAROA protocol version implemented by the IS server.

⟨ **SRVPROTOVER** ⟩

Possible replies from the IS are:

- In case of success:

⟨ **OK**<b>VERSION ⟩

where:

– VERSION: string containing the TAAROA protocol version implemented by the server.

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.16 UNREGPHYMACH – *Physical Machine Unregistration* request

Sent by a MM (or other clients) to the IS for unregistering a specific physical machine.

⟨ **UNREGPHYMACH**<b>PHY_ID ⟩

where:

- PHY_ID: integer number §5.1.1 representing the physical machine identifier.

Possible replies from the IS are:

- ⟨ **OK**<b>PHY_ID ⟩ in case of success, where PHY_ID is the integer identifier of the unregistered physical machine (the same received in the request message).

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

Side Effects: all virtual machines running on this machine MUST be unregistered as well.

5.2.17 UNREGREPO – *Repository Manager Unregistration* request

Sent by a RM (or other clients) to the IS for unregistering itself (a specific repository manager).

⟨ **UNREGREPO**<b>RM_ID ⟩

where:

- RM_ID: integer number §5.1.1 representing the repository manager identifier.

Possible replies from the IS are:

- ⟨ **OK**<b>RM_ID ⟩ in case of success, where RM_ID is the integer identifier of the unregistered repository manager (the same received in the request message).

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

Side Effects: all services and related virtual machines registered by this RM MUST be unregistered as well.

5.2.18 UNREGSERV – *Service Unregistration* request

Sent by a RM to the IS for unregistering a specific service.

⟨ **UNREGSERV**<b>S_ID ⟩

where:

- S_ID: integer number §5.1.1 representing the service identifier.

Possible replies from the IS are:

- ⟨ **OK**<b>S_ID ⟩ in case of success, where S_ID is the integer identifier of the unregistered service (the same received in the request message).

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

Side Effects: all virtual machines associated to this service MUST be unregistered as well.

5.2.19 UNREGVM – *Virtual Machine Unregistration* request

Sent by a MM to the IS for unregistering a specific running service (virtual machine).

⟨ **UNREGVM**<b>VM_ID ⟩

where:

- VM_ID: integer number §5.1.1 representing the virtual machine identifier.

Possible replies from the IS are:

- ⟨ **OK**<b>VM_ID ⟩ in case of success, where VM_ID is the integer identifier of the unregistered virtual machine (the same received in the request message).

- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.2.20 UPDATEVMSTATUS – *Virtual Machine Status Update* request

Sent by a RM to the IS when it wants to update the execution status of a given submitted service (virtual machine).

⟨ **UPDATEVMSTATUS**<b>VM_ID<b>STATUS ⟩

where:

- VM_ID: integer number §5.1.1 containing the submitted service (virtual machine) identifier.

- STATUS: integer number §5.1.1 representing the execution status of the submitted service (§3.1.3).

Possible replies from the IS are:

- ⟨ **OK**<b>STATUS ⟩ in case of success, where STATUS is the new execution status of the submitted service (virtual machine).
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.3 Messages issued to the Repository Manager

5.3.1 SRVPROTOVER – *Protocol Version* request

Sent by a client to the RM for getting information about the TAAROA protocol version implemented by the RM server.

⟨ **SRVPROTOVER** ⟩

Possible replies from the RM are:

- In case of success:

  ⟨ **OK**<b>VERSION ⟩

  where:

  – VERSION: string containing the TAAROA protocol version implemented by the server.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.3.2 STOPVM – *Service Stop* request

Sent by a SC to the RM for stopping a given submitted service (virtual machine).

⟨ **STOPVM**<b>VM_ID ⟩

where:

- VM_ID: integer number §5.1.1 representing the submitted service (virtual machine) identifier.

Possible replies from the RM are:

- ⟨ **OK**<b>VM_ID ⟩ in case of success, where VM_ID is the identifier of the stopped submitted service (virtual machine).
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.3.3 SUBMITVM – *Service Submission* request

Sent by a SC to the RM for submitting a given service.

⟨ **SUBMITVM**<b>S_ID<b>PHY_ID ⟩

where:

- S_ID: integer number §5.1.1 representing the service identifier.
- PHY_ID: integer number §5.1.1 representing the identifier of the physical machine where the service has to be executed.

Possible replies from the RM are:

- ⟨ **OK**<b>VM_ID ⟩ in case of success, where VM_ID is the integer identifier of the virtual machine where the submitted service (virtual machine) is running.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.4 Messages issued to the Scheduler

5.4.1 SRVPROTOVER – *Protocol Version* request

Sent by a client to the SC for getting information about the TAAROA protocol version implemented by the SC server.

⟨ **SRVPROTOVER** ⟩

Possible replies from the SC are:

- In case of success:

  ⟨ **OK**<b>VERSION ⟩

  where:

  – VERSION: string containing the TAAROA protocol version implemented by the server.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.4.2 STOPSERV – *Service Stop* request

Sent by a TC to the SC for stopping a given submitted service (virtual machine).

⟨ **STOPSERV**<b>VM_ID ⟩

where:

- VM_ID: integer number §5.1.1 representing the identifier of a running service instance.

Possible replies from the SC are:

- ⟨ **OK**<b>VM_ID ⟩ in case of success, where VM_ID is the identifier of the stopped service instance.
- ⟨ **ERR**<b>CODE ⟩ otherwise, where CODE is an integer number representing an error code.

5.4.3 SUBMITSERV – *Service Submission* request

Sent by a TC to the SC for starting the execution of a given service.

⟨ **SUBMITSERV**`<b>S_ID` ⟩

where:

- `S_ID`: integer number §5.1.1 representing the service identifier.

Possible replies from the SC are:

- ⟨ **OK**`<b>VM_ID` ⟩ in case of success, where `VM_ID` is the integer identifier of the running instance of the given service.
- ⟨ **ERR**`<b>CODE` ⟩ otherwise, where `CODE` is an integer number representing an error code.

5.5 Messages issued to the Machine Manager

5.5.1 SRVPROTOVER – *Protocol Version* request

Sent by a client to the MM for getting information about the TAAROA protocol version implemented by the MM server.

⟨ **SRVPROTOVER** ⟩

Possible replies from the MM are:

- In case of success:

  ⟨ **OK**`<b>VERSION` ⟩

  where:

  – `VERSION`: string containing the TAAROA protocol version implemented by the server.
- ⟨ **ERR**`<b>CODE` ⟩ otherwise, where `CODE` is an integer number representing an error code.

5.5.2 STARTVM – *Virtual Machine Execution* request

Sent by a RM to the MM for starting a virtual machine given all the file necessary for running it.

⟨ **STARTVM**`<b>S_ID + <VM_IMAGE>` ⟩

where:

- `S_ID`: integer number §5.1.1 representing the service identifier.
- `<VM_IMAGE>`: all the file necessary for running the virtual machine.

Possible replies from the MM are:

- ⟨ **OK**`<b>VM_ID` ⟩ in case of success, where `VM_ID` is the integer identifier of the virtual machine where the submitted service (virtual machine) is running.
- ⟨ **ERR**`<b>CODE` ⟩ otherwise, where `CODE` is an integer number representing an error code.

5.5.3 STOPVM – *Virtual Machine Stop* request

Sent by a RM to the MM for stopping a given submitted service (virtual machine).

⟨ **STOPVM**`<b>VM_LOCAL_ID` ⟩

where:

- `VM_LOCAL_ID`: integer number §5.1.1 representing the submitted service (virtual machine) identifier.

Possible replies from the RM are:

- ⟨ **OK**`<b>0` ⟩ in case of success.
- ⟨ **ERR**`<b>CODE` ⟩ otherwise, where `CODE` is an integer number representing an error code.

5.6 Workflows Diagrams

To illustrate the communication protocol at work, we present in this section the two sample workflows previously described in §3.3. The type of modelling diagram used for introducing the interaction between the different TAAROA component is the UML *sequence diagram* [6]. This kind of diagram shows how components communicate with each other in terms of a sequence of messages. Furthermore, it indicates the lifespans of components relative to those messages.

Fig. 9 shows the sequence diagram for the service submission request corresponding to the Service submission workflow presented in §3.3.

Fig. 10 shows the sequence diagram for the service stopping request related to the Service stopping workflow described in §3.3.

6 Conclusions and Future Work

In this paper we presented the TAAROA middleware, a software system that tries to add the concept of service and Service Level Agreement (SLA) to the Grid computing paradigm, by using the virtualization technology. The current version of TAAROA has some limitations. The most important of these are the lack of a logic for mapping a high-level SLA specification to a low-level resource allocation and, as a consequence, the absence of a scheduling heuristic that properly assigns a Physical Machine to a Virtual Machine taking into account the preservation of SLA

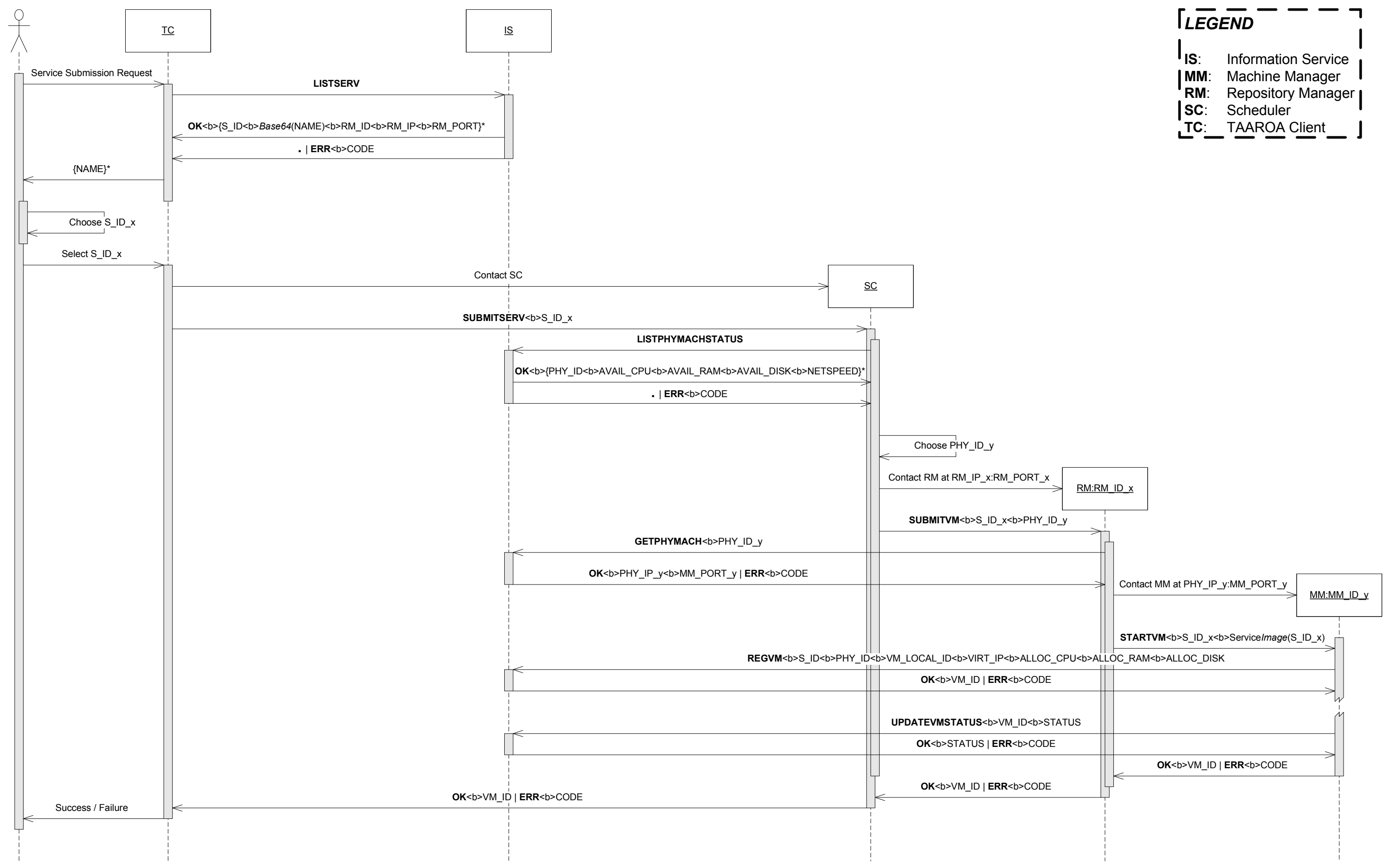

TC
IS
LEGEND
IS: Information Service
MM: Machine Manager
RM: Repository Manager
SC: Scheduler
TC: TAAROA Client
Service Submission Request
LISTSERV
OK<b>{S_ID<b>Base64(NAME)<b>RM_ID<b>RM_IP<b>RM_PORT}*
. | ERR<b>CODE
{NAME}*
Choose S_ID_x
Select S_ID_x
Contact SC
SC
SUBMITSERV<b>S_ID_x
LISTPHYMACHSTATUS
OK<b>{PHY_ID<b>AVAIL_CPU<b>AVAIL_RAM<b>AVAIL_DISK<b>NETSPEED}*
. | ERR<b>CODE
Choose PHY_ID_y
Contact RM at RM_IP_x:RM_PORT_x
RM:RM_ID_x
SUBMITVM<b>S_ID_x<b>PHY_ID_y
GETPHYMACH<b>PHY_ID_y
OK<b>PHY_IP_y<b>MM_PORT_y | ERR<b>CODE
Contact MM at PHY_IP_y:MM_PORT_y
MM:MM_ID_y
STARTVM<b>S_ID_x<b>ServiceImage(S_ID_x)
REGVM<b>S_ID<b>PHY_ID<b>VM_LOCAL_ID<b>VIRT_IP<b>ALLOC_CPU<b>ALLOC_RAM<b>ALLOC_DISK
OK<b>VM_ID | ERR<b>CODE
UPDATEVMSTATUS<b>VM_ID<b>STATUS
OK<b>STATUS | ERR<b>CODE
OK<b>VM_ID | ERR<b>CODE
OK<b>VM_ID | ERR<b>CODE
OK<b>VM_ID | ERR<b>CODE
Success / Failure

Figure 9. The sequence diagram for TAAROA service submission.

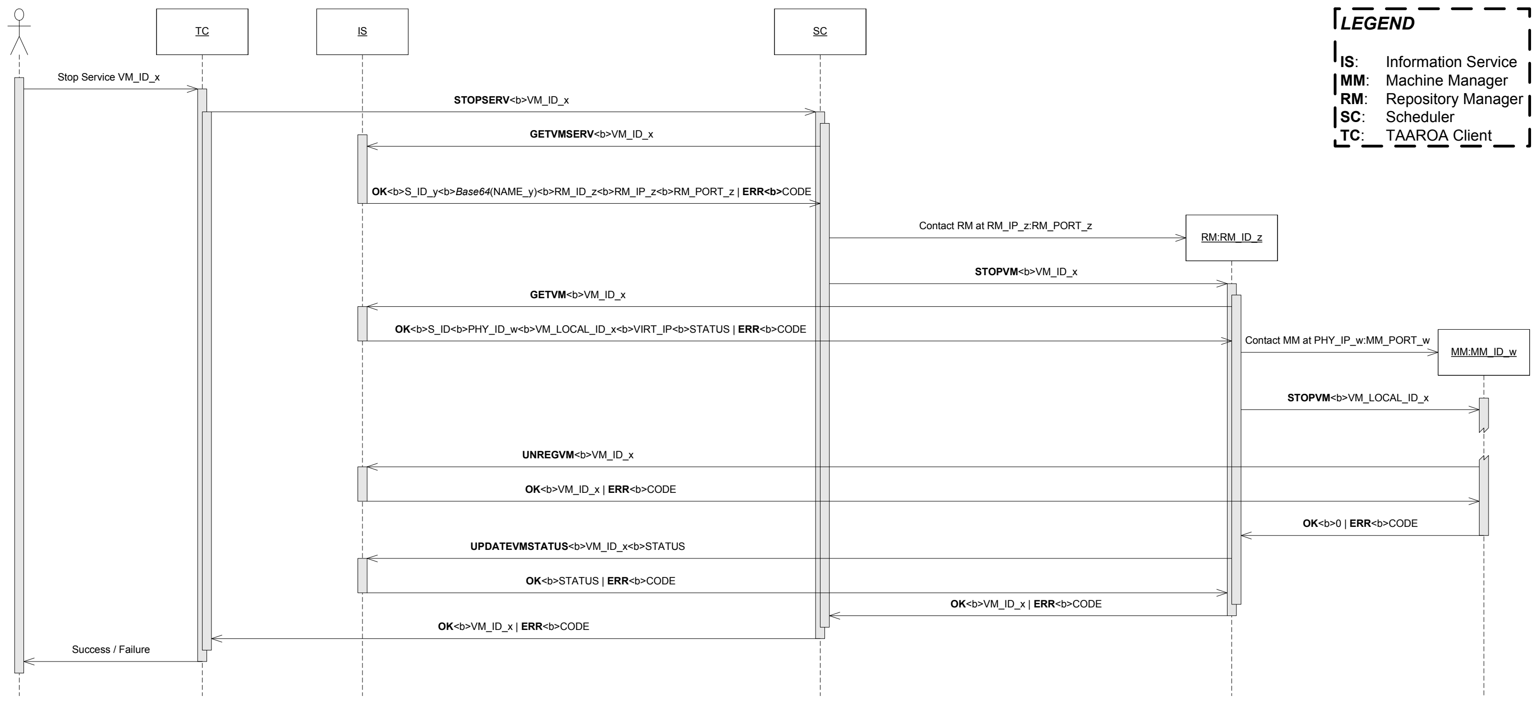


Figure 10. The sequence diagram for TAAROA service stopping.

constraints. This means that actually TAAROA is only able to provide a best-effort service: each service is scheduled for execution with a First-Come-First-Served policy and is assigned to the first available Physical Machine. In the future, we plan to provide a better support for proactively or reactively avoiding SLA violations, by creating specific performance models and exploiting, for instance, the Virtual Machines migration.

References

[1] C. Abrams. Service-oriented business applications break down barriers. Research Note AV-22-1413, Gartner Research, February 2004.

[2] N. Bieberstein, S. Bose, L. Walker, and A. Lynch. Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. *IBM System Journal*, 44(4):691–708, 2005.

[3] P. P.-S. Chen. The Entity-Relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[4] L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana, and G. Rackham. Impact of service orientation at the business level. *IBM System Journal*, 44(4):653–668, 2005.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, January 1979.

[6] O. M. Group. Unified Modeling Language: Superstructure v2.1.2. Specification formal/2007-11-02, OMG, November 2007.

[7] P. Liegl. The strategic impact of service oriented architectures. In *Proc. of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pages 475–484, Los Alamitos, CA, USA, 2007. IEEE Computer Society.