



Cross-Layer and Multi-Objective Programming Approach for
Next Generation Heterogeneous Parallel Computing Systems

Project Number 688146

D5.2 – Integrated reference system

**Version 1.0
2 April 2019
Final**

Public Distribution

**University of York, Easy Global Market, GMV, Intecs,
The Open Group, University of Stuttgart,
Unparallel Innovation, WINGS ICT Solutions**

**Project Partners: Easy Global Market, GMV, Intecs, The Open Group, University of Stuttgart,
University of York, Unparallel Innovation, WINGS ICT Solutions**

Every effort has been made to ensure that all statements and information contained herein are accurate, however the PHANTOM Project Partners accept no liability for any error or omission in the same.

© 2019 Copyright in this document remains vested in the PHANTOM Project Partners.

PROJECT PARTNER CONTACT INFORMATION

Easy Global Market Philippe Cousin 2000 Route des Lucioles Les Algorithmes Batiment A 06901 Sophia Antipolis France Tel: +33 6804 79513 E-mail: philippe.cousin@eglobalmark.com	GMV José Neves Av. D. João II, Nº 43 Torre Fernão de Magalhães, 7º 1998 - 025 Lisbon Portugal Tel. +351 21 382 93 66 E-mail: jose.neves@gmv.com
Intecs Silvia Mazzini Via Umberto Forti 5 Loc. Montacchiello 56121 Pisa Italy Phone: +39 050 9657 513 E-mail: silvia.mazzini@intecs.it	The Open Group Scott Hansen Rond Point Schuman 6 5 th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of Stuttgart Bastian Koller Nobelstrasse 19 70569 Stuttgart Germany Tel: +49 711 68565891 E-mail: koller@hlrs.de	University of York Neil Audsley Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325571 E-mail: neil.audsley@cs.york.ac.uk
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	WINGS ICT Solutions Panagiotis Vlacheas 336 Syggrou Avenue 17673 Athens Greece Tel: +30 211 012 5223 E-mail: panvlah@wings-ict-solutions.eu

DOCUMENT CONTROL

Version	Status	Date
0.7	Initial release for partner review	25 February 2019
0.8	Updates according to component integration results	29 March 2019
0.9	Further updates and editing	1 April 2019
1.0	Final Release	2 April 2019

TABLE OF CONTENTS

1. Introduction.....	1
1.1 Objectives.....	1
1.2 Platform Overview.....	1
2. Inter-Component Communication: Repository.....	4
2.1 Overview.....	4
2.2 Interface	5
2.2.1 Queries	5
2.2.2 Notification Mechanism.....	6
2.3 Security integration	7
2.4 Infrastructure for testing the Inter-Component Communication.....	10
3. Integrated reference System.....	12
3.1 Description	12
3.2 Implementation	15
3.3 Installation and Usage guide.....	16
4. Requirements Assessment	20
5. Conclusions	27

EXECUTIVE SUMMARY

This deliverable presents the PHANTOM Framework as an integration of the several PHANTOM tools previously developed. It starts by presenting one of the key elements in the integration – the Repository – as it enables tools to exchange information between them and allow the implementation of a “wait-and-run” model where tools wait for the output of other tools before starting its activity. Moreover, that repository also enables the execution of the security mechanism provided by the Security Framework.

This deliverable also describes the information flow between the PHANTOM tools, through the different stages of the application analysis and execution implemented in a set of 2 virtual machines – for FPGA specific tools, and another for the remaining tools. It is also presented as set of scripts to help the user on the management of the virtual environments and on the configuration and execution of PHANTOM tools for a specific application.

At last, is shown in this deliverable the evaluation, for PHANTOM developers, of the requirements related with the overall design and capabilities of the PHANTOM framework. From a total of 55 requirements, 53 were considered ‘Fully’ met and 2 ‘Partially’ met

1. INTRODUCTION

1.1 OBJECTIVES

This deliverable aims to be used as manual for PHANTOM users to get an overview of the tools that compose the PHANTOM framework, how the several tools are organised and communicate among them to achieve PHANTOM objectives. In this context is presented and described a tool developed to support the communication and the activities of the other components – the Repository.

This deliverable also presents the PHANTOM Integrated Platform, describing the collection of tools developed to support the configuration and usage of PHANTOM tools. Alongside to these tools, this deliverable introduces a virtualised system where the PHANTOM tools, and its dependencies, are deployed and configured. Such system provides newcomers to PHANTOM framework with ready-to-use environment where they can get a first impression of the framework.

As this document describes the collection of PHANTOM tools as an integrated framework, in this deliverable is also presented the final evaluation of the requirements related with the PHANTOM framework design.

1.2 PLATFORM OVERVIEW

A brief introduction to PHANTOM architecture is needed to better understand the role of each PHANTOM tool on the overall PHANTOM framework. Figure 1 presents an overview of the flow between the tools that compose the PHANTOM platform. The top of the figure shows the *Design* stage, where the application is analysed, and deployment strategies are devised. This stage starts with the user's inputs, labelled as the *Programming Model*. These inputs are analysed, and the application components' code is restructured, appropriately annotated, and later parallelized by the tools. This is followed by the development of an optimized deployment plan depending on the user's requirements.

The lower part of the figure shows the components' interaction during the runtime of *phantomized* applications.

The following components are briefly described as the understanding of their objectives and roles is important to better understand the following sections.

Programming Model

The programming model leverages a functionally-oriented component-based methodology to the application design. The developer structures their application as an interconnected network of communicating software components, each of which follows specific guidelines. Along with the application source code, the user provides an initial deployment of their application, which defines where the components of the application may be located in the target hardware. This may be underspecified, allowing the platform to optimise the deployment according to monitoring and testing results. The developer

also defines non-functional requirements of the components, in terms of response time, power usage, security, or many other application-specific metrics.

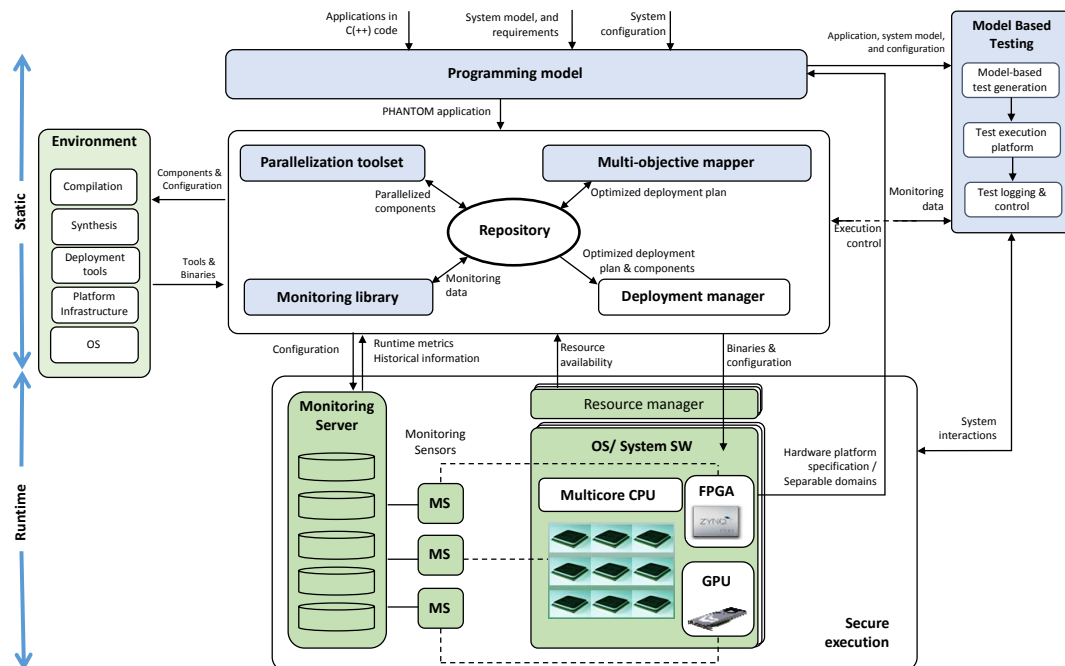


Figure 1: PHANTOM architecture

Parallelisation Toolset

This tool analyses the component's source code and transforms it according to parallelization directives for each of the targeted hardware platform's abilities. The current version of the Parallelization Toolset design concentrates on two main functionalities: Component Parallelization and Parallelization Technique Selection. Parallelization can be achieved through injection of parallelization technology (i.e. OpenMP, CUDA) annotations and generation of IP Cores, aiming to the specific platform that the components are executed on.

Repository

The PHANTOM Repository is the integration data layer that can store the user's components and serves all communication purposes between the user and the PHANTOM tools as well as within the latter by endorsing RESTful protocols.

Multi-Objective Mapper

The Multi-Objective Mapper (MOM) has the task of defining an optimal deployment plan for a PHANTOM application. This implies that MOM needs to take decisions on the mapping of the parallelizable components and on the shared data communications throughout the target hardware architectures (CPU, GPU, FPGA).

Monitoring Framework

The PHANTOM monitoring framework supports the collection and storage of monitored metrics based on hardware availabilities and platform configuration. The target platform of the monitoring client includes CPUs, GPUs, ACME power measurement kit and FPGA-based platform.

Deployment Manager

The Deployment Manager is responsible for the final code generation / refactoring stage which adds the communication and consistency code that is required by a given deployment as well as building different parts of the project on the selected hardware platforms specified by the Optimized Deployment Plan. Specifically, the Deployment Manager generates the necessary files for the communication among components by using the corresponding techniques (MPI and POSIX functions). Furthermore, the DM generates the scripts that become available to the user for the compilation, linking, and deployment of the binaries on the hardware infrastructure (CPU, GPU, FPGA).

Model-Based Testing

Model-based Testing (MBT) is used to carry out early validation and black box testing for use case applications on the PHANTOM platform focusing global functional and non-functional properties. It reads design specifications, including functional and non-functional requirements, system behaviour descriptions and PHANTOM network component descriptions

During early validation, the outputs are functional validation results and performance estimation results. In test execution phase, the outputs are functional and non-functional testing verdicts indicating if test cases pass or fail.

More details about the PHANTOM architecture can be found in Deliverable D1.3 - *Enhanced design for Cross-layer Programming, Security, and Runtime monitoring*.

2. INTER-COMPONENT COMMUNICATION: REPOSITORY

2.1 OVERVIEW

The PHANTOM Repository is an integration data layer that supports the exchange of information and files between PHANTOM tools, as well as between PHANTOM tools and the users. In particular, the Repository is in charge of store the user's components (including alternate versions of those components generated by the PHANTOM Parallelisation Toolkit) and serves all communication purposes between the user and the PHANTOM tools as well as within the latter by endorsing RESTful protocols. Thus, the Repository allows both the users and the platform tools to abstract from the distributed and heterogeneous nature of the targeted hardware and serves a one-step communication hub for all inter-platform communications.

The Repository has a clearly defined interface for storing and accessing the different types of content to facilitate its use by different tools, which is described later in this section.

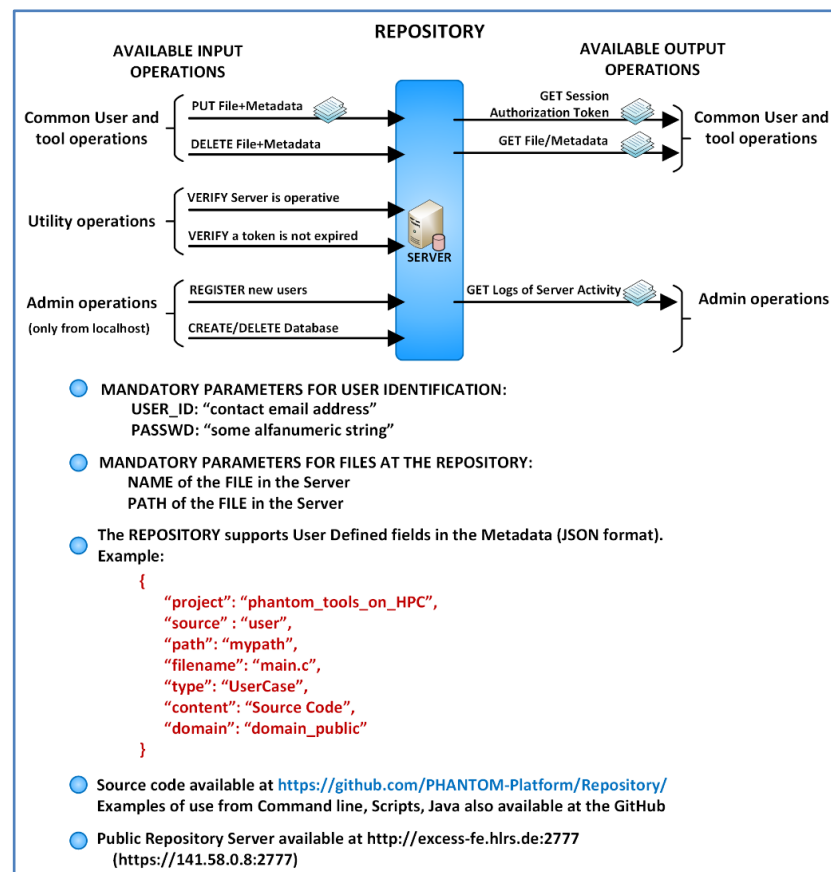


Figure 2. PHANTOM Repository interface between PHANTOM tools, as well as between the latter and the users, storing files and metadata.

Among the contents to be stored in the repository, we can highlight the programming model provided by the developer, the test models and their results generated by MBT,

different versions of the generated code, deployment plan, and execution results such as security logs and monitoring metrics.

In addition to data storage, the Repository will also provide a pub/sub (publish-subscribe) mechanism for PHANTOM components to subscribe to data and be notified when certain project folder or file is updated. This is useful in scenarios such as PHANTOM users trigger the execution of an application when given input data, and users are notified when the execution is over and output data is ready.

2.2 INTERFACE

2.2.1 Queries

This section lists the different operations available to interact with the Repository. Those operations are read/write/list files and retrieve a zip file of a folder or single file. For each one of the operations are shown examples and the possible responses. The examples of the HTTP requests appear with underlined font, and the reply from the server in italic font:

- **Example of Testing the USER PERMISSIONS on different domains:**

Test read permission of a registered user in the public domain (domain_public):

```
curl -XGET http://localhost:8000/permission?user=montanana@hlrs.de&domain=domain_public&access=r;
```

The expected response is (user_id must be present on the access policy):

200: Access granted

Test write permission of a registered user in a domain where has granted such permission:

```
curl -XGET http://localhost:8000/permission?user=montanana@hlrs.de&domain=domain_hlrs&access=w;
```

The expected response is:

200: Access granted

Test write permission of a registered user in a domain where has NOT granted such permission:

```
curl -XGET http://localhost:8000/permission?user=montanana@hlrs.de&domain=domain_gmv&access=w;
```

The expected response is:

403: Access denied

Notice that any USER NOT REGISTERED in the security-policy will get a REJECTION for any requested access.

- **Example of query the domain of a single file:**

Query for the domain of a file in the Repository:

```
curl -s -XGET http://localhost:8000/test_metadata?project=phantom tools on HPC\&source=user\&Path=
mypath%2F&filename=main.c
```

Example of possible response:

The domain of the file is: domain_hlrs

- **Example of obtaining a new Token:**

The other implemented functionalities require the use of a token as a way to indicate that a user has successfully been identified and authenticated. To obtain a token, a user must present valid credentials, consisting of a public user name and a secret password. The credentials presented are checked against a table of stored credentials. Following is an example of requesting a new token for a particular user:

```
curl -H "Content-Type: text/plain" -s -XGET
http://localhost:8000/login?email="montana@abc.com"\&pw="XXX" --output token.txt;
mytoken=`cat token.txt`;
```

This operation will store the Token, which consists on a text string, into the variable mytoken. The token serves as evidence that the requester has been authenticated, and is used in subsequent requests to the Repository to access resources.

- **Example of Downloading a file from the Repository:**

Example of download request, parameters specify the file to be accessed:

```
curl -s -H "Authorization: OAuth ${mytoken}" -H "Content-Type: multipart/form-data" -XGET
http://localhost:8000/test_download?project=phantom tools on HPC\&source=user\&filepath=mypath\&file
name=main.c ;
```

The expected response when the access is granted:

200: Access granted

The expected response when the access is denied:

403: Access denied

2.2.2 Notification Mechanism

This Repository allows users and tools to subscribe by using web sockets to get notifications of changes to files on a project or to a particular folder. Such notifications consist of forwarding a copy of the metadata JSON of the upload/updated files to only those who have subscribed

In particular, the subscription for updates is done by establishing a websocket connection (`ws://repository_address:repository_port/`) and sending a request message in JSON format, for instance for subscribing on updates on a particular project, or subscribe for such project but only on particular source:

```
{“user”: “bob@abc.com”, “project”: “demo_hpc”}
```

Or

```
{“user”: “bob@abc.com”, “project”: “demo_hpc”, “source”: “user”}
```

The first field is the user id of the subscriber, which currently is used for debugging purposes.

The notification message when a file is modified or uploaded consists on the metadata in JSON format of such file, as for example:

```
{“project”: “demo_hpc”, “source”: “user”, “path”: “folder_one”, “filename”: “main.c”,  
  “domain”: “domain_public”}
```

On the GitHub are uploaded examples in Java and Python of applications that subscribe to the updates of some project, and some type of source updates.

2.3 SECURITY INTEGRATION

The permissions-domain is defined for each file in their associated metadata that is provided in JSON format, and then files associated with the same project can be defined public or private to the user group independently of the other files in the same project.

The field in the metadata which defines the access domain is named “*domain*”. The value of this field can be “*domain_public*” when there is not any restriction on their access, or another valid value defined on the uploaded access policy loaded into the Security Server. More details about the specification of access policies are provided in the Security Server section.

The Figure 3 shows an example of two different projects, and their files and folders uploaded by users or different PHANTOM tools. The figure shows also the metadata of two of those files, where one has public access while the other has restricted access to only the users which belong to certain domain (as “*domain_hlrs*” in the figure).

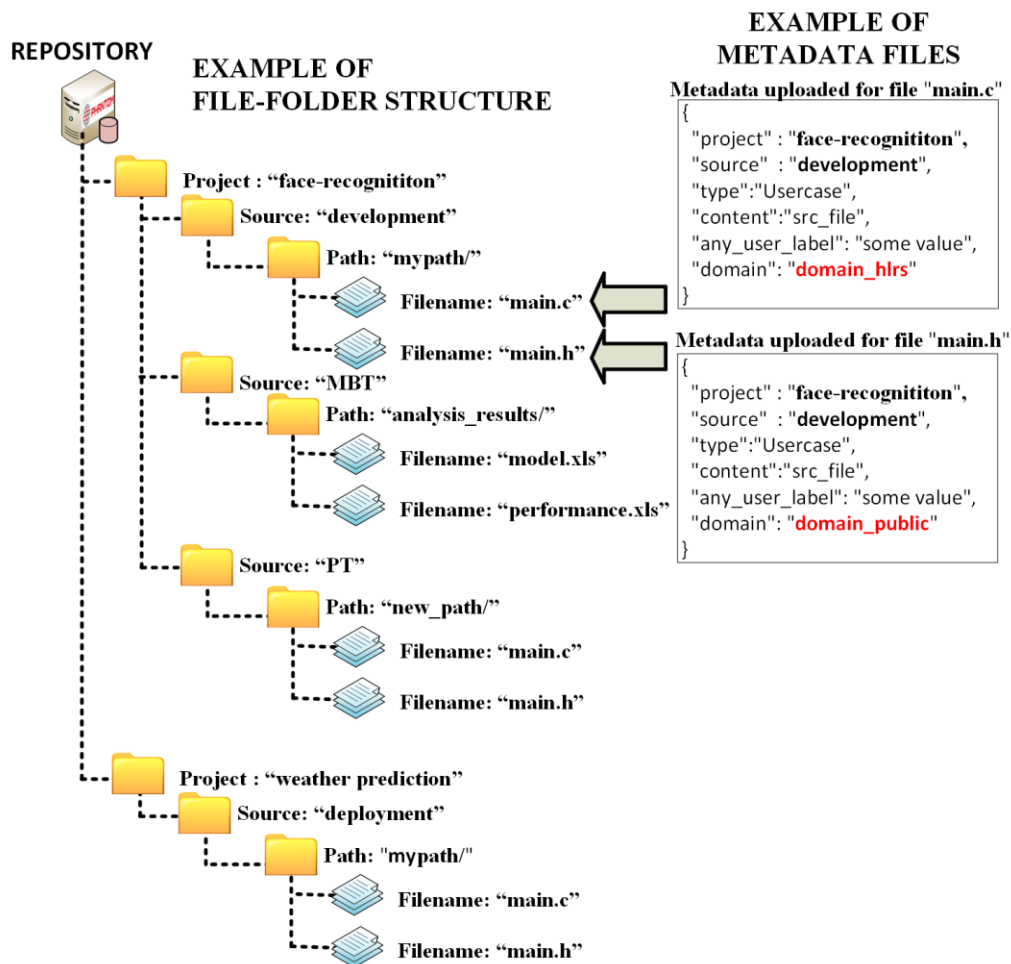


Figure 3. Example of metadata of two files on the same project. Each metadata in the figure defines a different security domain.

The Figure 4 shows the interaction between the Repository and the NGAC-policy server. In the figure, there are three different actors, first the developer (or user) and the PHANTOM tools, second the Repository, and third the NGAC-Policy server. This figure shows the interactions among them that are performed for every access request, which consist of seven steps.

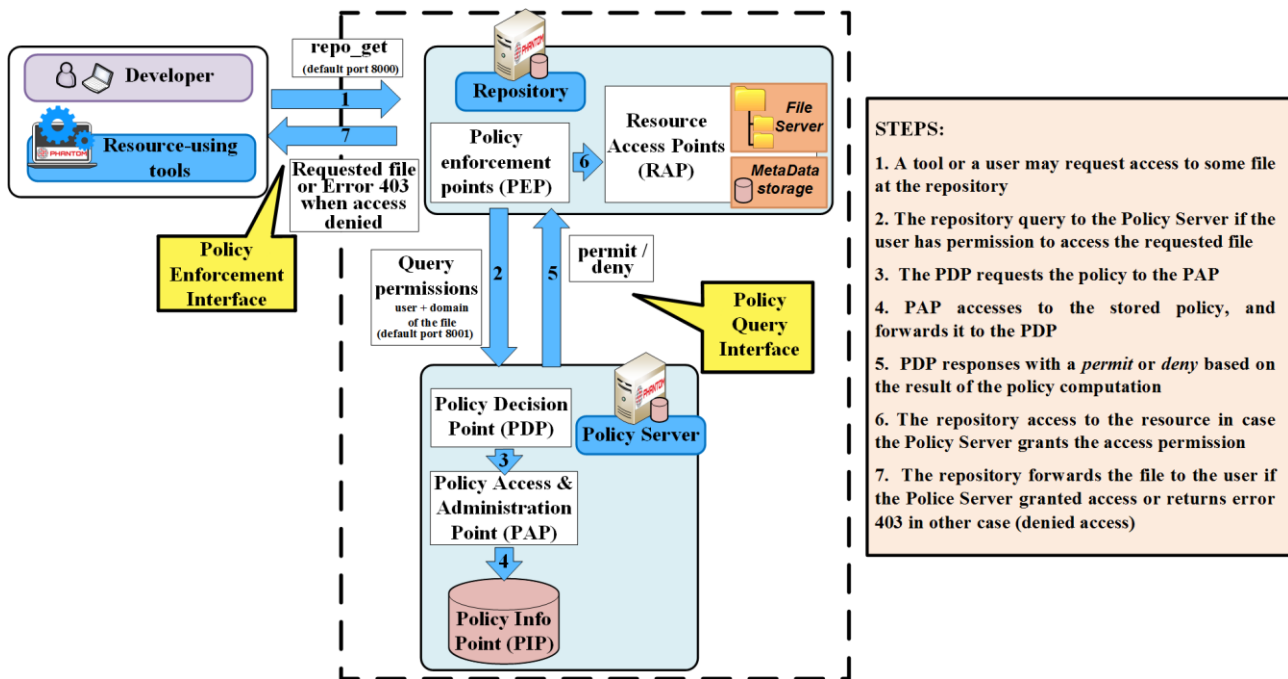


Figure 4. Description of the resource access.

Here is described the interaction of the Repository with the Security server. The Figure doesn't show the Policy Server as part of the Repository because both run independently. In particular, other PHANTOM tools can act as Policy Enforcement Points (PEPs) and interact with the Policy Server.

The process starts with the first step (1) when a read or write access request is made to the Policy Enforcement Interface of the repository. With such a request, the user or tool provides a token¹ as an authorization key.

In the second step (2), the Repository acts as Policy Enforcement Point (PEP) identifying the associated user_id if the token is valid and requesting to the Policy Query Interface of the Policy Server whether the user is authorized to access the requested object. In particular, the access request is characterized by the user (or tool) on behalf of whom the request is being made, the object that is to be accessed, and the operation that is to be performed on the object by the access.

(3) The Policy Decision Point (PDP) computes the decision based on the received request and the defined access policy. For such purpose, it requires the current policy through the Policy Access Point (PAP).

(4) Policy Access Point (PAP) access to the current policy stored in the Policy Information Point (PIP). The policy (or policies) in the PIP was previously loaded into the PIP during initialization of the Server from a file containing an attribute-based

¹ A user can obtain as many different tokens as may be needed. Each token is a unique text string that will be used as evidence that the user has previously been successfully identified and authenticated. Each token has associated some additional values as the valid time interval for its use.

security policy specification represented in a declarative language. The details of the policy specification framework and language are presented elsewhere. Policies are prepared for use by the Policy Server using a Policy Tool that enables a security administrator to develop and test the policy without interacting with the policy server.

(5) It is the policy that determines whether the identified user is permitted to perform the indicated operation on the named object. Based on the result of the policy computation, the PDP will respond to the access request with “permit” or “deny” (5).

In the step (6), based on the reply from the PDP, the Repository PEP performs a read or write access (according to the initial request) to the requested object.

On the final step (7), the Repository Server will return a success or failure message and the requested object in the case of a read operation.

2.4 INFRASTRUCTURE FOR TESTING THE INTER-COMPONENT COMMUNICATION

Here, we provide a summarized description of the Infrastructure used for testing the Inter-Component communication.

In order to facilitate the developing and debugging of the inter-component communication. It was considered to provide online-uninterrupted service of the different Phantom Servers and Managers. The Phantom Servers and Managers were installed in an infrastructure accessible on internet at the IP address <http://141.58.0.8>. (Notice that ports are modified from the default values to: Repository at port 2777, App-Manager at port 2778, Monitoring Server at port 2779, Resource manager at port 2780, and Execution Manager at port 2781).

The Figure 5 shows the structure infrastructure, located at the HLRS centre. In particular, the PHANTOM Servers and Managers were installed in the computing node labelled as “BackEnd”. While, the Node01 is used for running PHANTOM applications.

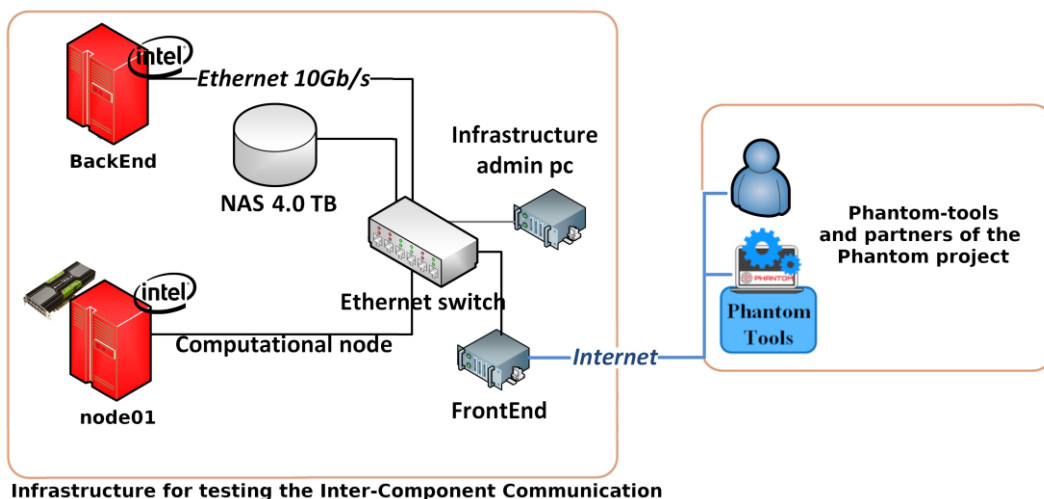


Figure 5. Infrastructure for testing the Inter-Component Communication.

The different services are available on different network ports, which are defined based on the network protocol of the HLRS centre and the currently available ones, which made the network-ports to be:

Phantom Servers and Managers					
	Repository	App Manager	Exec Manager	Resource Manager	Monitoring Server
Port	2777	2778	2781	2780	2779

Next, we provide a description of the characteristics of the components of the elements in the Figure 5.

BackEnd Machine (here run the PHANTOM Servers and Managers):

- 1 × Chenbro 4U 17.5" Compact Industrial Server Chassis RM42300
- 1 × Gigabyte® Server Board - MD70-HB0 (Rev. 1,2)
- 2 × Intel Xeon CPU: E5-2680 v3 - 12 cores, 30MB L3 smart cache, 2.5 GHz, 4 mem. ch. DDR4 2133 MHz
- 8 × SAMSUNG DRAM 16GB Samsung DDR4-M393A2G40DB0-CPB - 2133 MHz, CL15 - Registered DIMM - Dual Rank - ECC
- 1 × GPU: Nvidia TeslaK80 - 2x Kepler GK210; GPU Clock: 560-875 MHz; Shading Units: 4992; GDDR5 24 GB; 480 GB/sec; PCIe3.0 8GT/s x16
- 1 × Intel® Ethernet Server Adapter I350-T2 - PCI Express 2.1 x4 Low Profile - 1000Base-T x 2
- 1 × Hard disk: 500GB WD5003AZEX Black;
- 1 × SSD: 240GB Vertex460A

Node01 (for running phantom applications):

- 1 × Intel Server Chassis P4308XXMHGC
- 1 × Intel Server Board S2600COE
- 2 × Intel Xeon CPU: E5-2690 v2 - 10 cores; 25MB L3 smart cache; 3.0 GHz, 4 mem ch. DDR3 1866 MHz
- 8 × Hynix Memory 4 GB DDR3 HMT351U7EFR8C-RD - 1866 MHz PC3-14900, CL13 - Dual Rank - ECC
- 1 × GPU TeslaK40c - GPU Clock: 745 MHz; Shading Units: 2880; GDDR5 12288 MB; 288 GB/s; PCIe3.0 8GT/s x16
- 1 × Hard disk: 500GB WD5003AZEX Black; 1 × SSD: 128GB Vertex450

Ethernet Switch (interconnects the elements located at HLRS):

- 1 × Cisco Catalyst 3750X-24T-S, 24 10/100/1000

Frontend machine (forwards traffic from internet to the different computing nodes):

- 2 × Intel® Xeon® E5-2609 v2, 2.50 GHz, 4-Core, 10MB Cache
- 8 × 8 GB DDR3
- 4 × 1TB SATA3-HDD Seagate Constellation ES.3

Network Storage System (NAS):

- 1 × Intel® Xeon® E3-1220 v3 3100MHz 8MB Cache 4Core
- 2 × 8GB DDR3
- 2 × 2TB WD Caviar Red NAS HDD 64MB

3. INTEGRATED REFERENCE SYSTEM

The Integrated Reference System consists in a system where most of the PHANTOM tools are deployed and configured to communicate with each other. This system provides users of the PHANTOM framework (USERS) with a generic deployment of tools that allow them to test most of the functionalities provided by PHANTOM.

3.1 DESCRIPTION

The set of PHANTOM tools communicate with each other following a defined communication flow where outputs of a process of a specific tool can trigger a process on another tool. The communication and coordination between the PHANTOM tools are enabled and supported by a set of servers:

- Repository, supports the transfer of data (inputs and outputs) between PHANTOM tools and with the USER;
- Application manager, allows tools to report its execution states (*waiting* for inputs, *running* or *finished*), allowing other tools and the USER to evaluate the status of each tool; and
- Execution manager, is used to register executions intents and to track that state of the execution (*pending*, *compiling*, *running*, *completed*, and *rejected*).

PHANTOM tools can submit subscriptions to the servers and receive notifications when files are updated/created, or when the state of a tool or execution is updated. This subscription/notification system consists in the default control mechanism for execution control of PHANTOM tools, aiming to reduce the amount of effort required from the USER as it removes the need for control the individual execution of each tool. The communication flow between PHANTOM components and with the USER is described in Figure 6.

The flow starts with the PHANTOM USER uploading its application to the Repository (1a) and performing the corresponding configuration of PHANTOM tools to subscribe the processing of the application. This action creates a new entry in the application manager (1b) required to enable tracking the progress of PHANTOM tools for that application.

The first stage of the PHANTOM execution is called Static Analysis, corresponding to the analysis of the application code and of the component network description. This stage corresponds to execution of the Parallelization Toolset (PT) and the Model-based Testing (MBT). Regarding to the Parallelization Toolset, in this phase is executed the Code Analysis process (2a), where the code of PHANTOM application is analysed and are identified loops and functions that can be parallelised, and the component network is updated with parallelisation directives. In this step, it can also be started the execution of the MBT testing process through the realisation of the early validations (2b). These early validations are performed over the component network, which combined with the results from previous executions (if available) allow to calculate some metrics that can

help Generic Multi-Objective Mapper in the finding of mappings that best satisfy the user requirements.

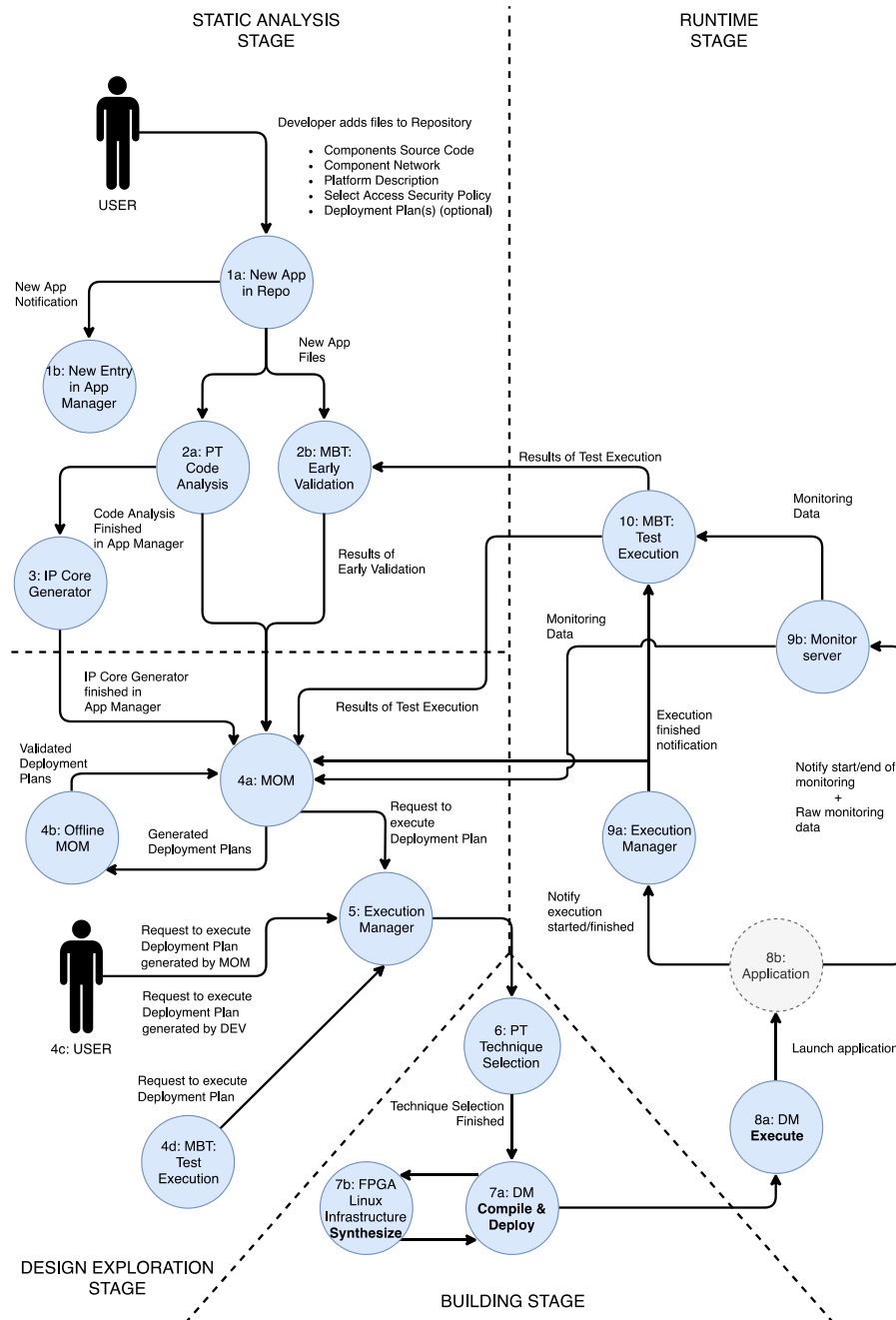


Figure 6 - PHANTOM tools workflow

If the Code Analysis finds pragmas, defined by the USER, to implement a specific function for a FPGA target, then Code Analysis signals on the Component Network's description that the corresponding component should have a FPGA implementation. After the conclusion of the Code Analysis activities, and corresponding indication in the Application Manager, the IPCore Generator (3) receives the corresponding notification

and analyses the produced Component Network. If a component has FPGA as target and no specific implementation is identified, then the IPCore Generator analyses the code of that component and attempts to generate the FPGA implementation of the function identified by the USER. In case of success, the FPGA related artefacts are uploaded to the Repository and the Component Network is updated with the corresponding paths.

In the next step, the Generic Multi-Objective Mapper (MOM) (4a), after receiving the notification of the conclusion of Code Analysis, IPCore Generator and MBT, analyses the component network and the platform description files to create and refine multiple deployment plans for deploying the application's components across the available hardware. This deployment plans will be optimised to better meet the USER's objectives and requirements.

After being generated, the deployment plans are analysed by the Offline MOM (4b) in order to perform an initial check if the real-time constraints can be satisfied using the selected mapping. This validation aims to exclude deployments that, due to the hardware specification, will not be able to satisfy the real-time requirements. However, this validation does not provide any confirmation about the real fulfilment of the real-time requirements as it must be verified through the execution of the application. After the identification of the valid deployment plans, these are uploaded to the repository and an execution request is submitted to the Execution manager for each deployment plan.

MOM has a cyclic behaviour, always waiting for the results of executions and MBT assessments to refined and create new deployment plans, being this stage called Design Exploration. As such, MOM status in the Application Manager will not assume the state of "finished" and will alternate between "waiting" when waiting for the other PHANTOM tools or for the results of the execution of applications, and "running" when generating new deployment plans.

When the Execution Manager receives an execution request (5), a new entry is created on the table of executions with the state "pending" and, by default, the Parallelisation Toolset – Technique Selection receives a notification and starts its execution. Possible other options are to wait for USER (4c) or MBT (4d) input to before starting the processing of an execution request.

The start of Technique Selection (6) activity represents the beginning of the *Building Stage*. This stage is represented with the execution request having the status "compiling". In this stage, parts of the component code are replaced to match the hardware and conditions and decisions specified on the deployment plan. When the Technique Selection finishes the refinement of the component's code, it updates the Application Manager with the "finished" status, which will trigger the execution of the Deployment Manager.

The Deployment Manager (7a) is responsible for preparing the application for the compilation and deployment processes. This preparation consists in the generation of the deployment scripts and Makefiles to provide the application with the required compilation dependencies. By using the information on the Hardware Description provided by the USER and the deployment plan associated to the execution, the

Deployment Manager attempts to connect to the specific computation resource and deploy the code and scripts, start the compilation of the application's components on the target machine and starts the execution of the application.

If the deployment plan specifies that some component should be deployed in a FPGA, then the Deployment Manager will call the FPGA Linux Infrastructure (7b). In this call, the Deployment Manager identifies the IP Core to be deployed and the address of the FPGA where it must be deployed. The FPGA Linux Infrastructure will use this information and include the IP Core in the build of a Linux Operating System that will be automatically be deployed on the target FPGA. After the installation of the bundle Linux OS + IP Core on the target FPGA, the Deployment Manager will automatically connect to the FPGA in order to compile and start the execution of the software part of the component deployed on the FPGA.

When all components are deployed and compiled in the corresponding target machines, the Building Stage is over and starts the Runtime Stage. In this stage the Deployment Manager starts the application (8a and 8b). When the application starts its execution, it updates the state of the corresponding execution request to “running” (9a) and signals the Monitoring Framework to start collection execution metrics (9b). In a similar way, when the application finishes its execution it will also inform the Execution Manager and the Monitoring Framework. The runtime metrics collect by the Monitoring Framework will be provide to MBT: Test Execution (10) to allow it to evaluation it the USER requirements are met in this execution. The results of MBT, as well as monitoring metrics are then feed into MOM, supporting its activity of Deployment Plans refinement, returning the application to the Design Exploration stage.

3.2 IMPLEMENTATION

The reference system consists in two Virtual Machines with all PHANTOM tools installed as well as the dependencies need to run each tool. The tools developed to work with FPGAs (IP Core Generator and FPGA Linux) are deployed in a virtual machine with Xilinx tools installed, to ease the license management and to reduce the image burden on the USER machine when the USER is not interested in PHANTOM FPGA capabilities –FPGA VM; and all the other tools and servers are deployed in the second machine – Main VM.

The Main VM is also the machine where are deployed a set of scripts, “User-Scripts”, developed to help the USER on the configuration and usage of PHANTOM tools, being the only VM and tools that the USER must interact directly with. These scripts are available on the project's GitHub².

Regarding to the technologies used to implement the VMs and User-Scripts, The Main VM is based in the VM provided By ROSE Compiler, a dependency of the Parallelization Toolset, a Ubuntu 16.04 image with all the dependencies installed to run deployed PHANTOM tools, plus OpenSSH and OpenMPI to allow the communication with other machines and to run local deployments of the USER application (if requested

² <https://github.com/PHANTOM-Platform/PHANTOM-User-Scripts>

by the USER). The FPGA VM is based on a Ubuntu 18.04 image with the dependencies required to run the FPGA tools plus openSSH to enable the communication with other machines.

The User-Script tools are implemented using Python and bash scripts, being very dependent on organisation of the filesystem and correct knowledge about location of each tool to perform its role. There are 2 files and 2 folders in the root directory of the User-Scripts that deserve special relevance:

- `settings.py` – this file is where the USER should do all the configurations related with its application. Configurations will be automatically propagated to all PHANTOM tools.
- `start-PHANTOM.py` – Script to be executed by the USER to start the analysis and execution of the USER application. This script uploads the application to the specified repositories and start PHANTOM tools with the required configurations to allow the automatic execution of the tool flow represented in Figure 6.
- `management-scripts` – this folder contains a set of scripts to ease the performance of some tasks over the deployed tools, like starting/stopping local servers, clean local servers' databases, and perform the update of the installed PHANTOM tools.
- `Templates` – this folder contains templates required for the generation of the configuration file of each tool. The User-Scripts will use the information in 'settings.py' and these templates to generate the configuration file required by each tool and deploy them on corresponding directory.

3.3 INSTALLATION AND USAGE GUIDE

The Virtual Machines are distributed in OVA format and can be imported with a virtualization client like VirtualBox³ or VMWare⁴.

For Main VM the minimum specifications tested were 2 CPUS and 4GB of RAM, while the FPGA VM was tested with 8 CPUS and 8GB of RAM. However, since the Xilinx SDK is, by far, the tool that requires more resource, deployment with fewer resources should also work, but at performance costs⁵.

Next will be presented the recommended steps for the correct usage of the Main VM:

1. **Update of the PHANTOM tools** – Is always recommended to run the script 'User-tools/management-scripts/update-tools.sh' to be sure that you are using the latest version of the PHANTOM tools. Please have in mind that if the update

³ <https://www.virtualbox.org>

⁴ <https://www.vmware.com/products/workstation-player.html>

⁵ https://www.xilinx.com/html_docs/xilinx2018_3/SDK_Doc/xsct/intro/xsct_system_requirements.html

updates the 'update-tools.sh' script, then the 'update-tools.sh' script must be executed again in order apply the changes. To execute the update script run:

```
demo@ubuntu:~/phantom-tools/User-tools/management-scripts$ bash update-tools.sh
```

Is worth to notice that this script starts the local servers upon exit.

2. **Start local servers (optional)** - If the USER wants to use local instance of the servers and did not run the update script, then the servers must be started manually by running the command:

```
demo@ubuntu:~/phantom-tools/User-tools/management-scripts$ bash start-servers.sh
```

3. **Copy of USER application to VM** - The next step consists in the deployment of the USER application code to be analysed inside the Virtual Machine at a location of the USER choice: \$CODE_DIR
4. **Configuration of 'settings.py'** – The file 'settings.py' must be configured to according to the characteristics of the application and the intentions of the USER. The 'settings.py' is divide in 3 main zones:

- i. **Repositories configurations** – used for the USER specify the location of the repositories to be used (localhost or remote location) and credentials. E.g.:

```
# Set the Repository IP address and port
#repository_ip = "141.58.0.8"
#repository_port = 2777
repository_ip = "localhost"
repository_port = 8000

# Authentication credentials
user =
password =
```

- ii. **Application configurations** – Used for the USER to indentify application specific properties:

Name of the application to be used server and by PHANTOM tools to identify the application:

```
app_name = "WINGStest3"
```

Path for the root of the application's folder (to upload makefile and cla.in)

```
root_path = "/home/demo/phantom-tools/Examples/WINGStest3"
```

Path for the folder with the source code

```
src_path = "/home/demo/phantom-tools/Examples/WINGStest3/src"
```


Path for the folder with description files (Component Network and Platform Description)

```
desc_path = "/home/demo/phantom-  
tools/Examples/WINGStest3/description"
```

Path for the folder with PHANTOM API files

```
phantom_path = "/home/demo/Desktop/phantom-  
tools/PHANTOM_FILES"
```

Link to the where the marketplace is hosted and name of the folder where IPCores should be stored locally

```
ipMarket_path = "https://github.com/PHANTOM-Platform/PHANTOM-  
IP-Core-Marketplace.git"  
ip_folder = "IPCore-MarketPlace"
```

Path for folder with application inputs

```
inputs_path = ""
```

Name of the component network file to be used

```
CompNetName = "cpn.xml"
```

Name of the platform description file to be used

```
PlatDesName = "hw_local.xml"
```

- iii. **Tools Configurations** – This section contains the parameters for configuring the PHANTOM tools. It includes the path for each tool deployed on the machine. E.g.:

```
# MOM location
```

```
MOM_path = "/home/demo/phantom-tools/GenericMOM"
```

And tool specific arguments. E.g.:

```
PT_mode = "on" #operation mode: on | off - "on" to run PT normally,  
"off" to skip code analysis process
```

In this section can also be found the property for the address of the FPGA VM, as well as the SSH port to be used

```
FPGAVM_ip = ""
```

```
FPGAVM_port =
```

5. **Run the start script** – The last step consists in the execution of script that will: upload all the needed files to the specified repositories; register the application on the Application manager; and configure and launch each tool. To start this script run:


```
demo@ubuntu:~/phantom-tools/User-tools$ ./start-PHANTOM.py
```

This command as several options as shown the when using the ‘-h’ flag:

```
demo@ubuntu:~/Desktop/phantom-tools/User-tools$ ./start-PHANTOM.py -h
usage: start-PHANTOM.py [-h] [-u] [-d] [-i] [-c] [-m] [-p]

Tool to support the execution of an application on PHANTOM Framework

optional arguments:
  -h, --help            show this help message and exit
  -u, --noUpload         Do not (re)upload the application to the repository.
                        (Application should be already in repository)
  -d, --onlyDesc         Only re-uploads the description files to the repository)
  -i, --skipInputs       Do not (re)upload the application inputs to the
                        repository. (Inputs should be already in repository)
  -c, --clean            Clean all the data in repositories and temporary cache
                        on PHANTOM tools. Automatically update PHANTOM_FILES
                        (-p)
  -m, --ipmarket         Uploads the IP Core Market place to the repository
  -p, --phantomfiles     Uploads the PHANTOM files (PHANTOM API and Monitoring
                        API)
```

During the execution of the application, the terminal windows are launched with the output of each tool. This way the USER can get a better understanding of what is happening and get feedback about any issue that may have occur with the execution of a tool. An example of an execution is shown in Figure 7, where the ‘./startPHANTOM’ script launched new terminals for the Parallelization Toolset (PT), Multi-Objective Mapper (MOM) and Deployment Manager (DM).

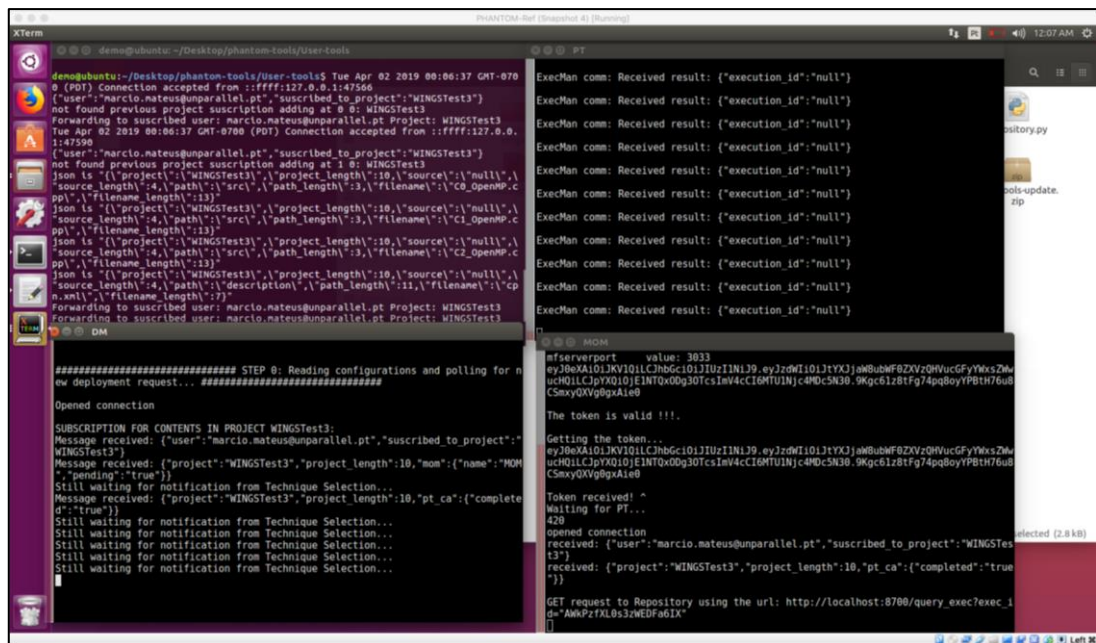


Figure 7 - Example of an execution of the start-PHANTOM.py script launching PT, MOM and DM

4. REQUIREMENTS ASSESSMENT

Parallelisation (General)			
ID	Description	Priority	Evaluation
U1	PHANTOM shall have a defined execution, memory and communications model	SHALL	Fully. PHANTOM programming model provides methods to define the execution communication and memory
U2	PHANTOM shall support uniform and non-uniform memory access models	SHALL	Fully. PHANTOM allows to describe UMA and NUMA systems and the shared memory API allows to abstract those systems
U5	The PHANTOM framework shall support multi-threaded concurrent tasks, including communication and synchronisation	SHALL	Fully. PHANTOM API allows synchronization and communication between concurrent tasks
U7	PHANTOM shall provide support for communications data-centric applications	SHALL	Fully. PHANTOM programming model allows the support of data-centric applications by specifying communication driven component network for each user application.
U8	PHANTOM shall support component-based application design	SHALL	Fully. PHANTOM applications are designed following a component-base approach

Heterogeneity of platform			
ID	Description	Priority	Evaluation
U16	The user shall be able to configure the target platform for a given project, which is composed by a set of supported target platforms	SHALL	Fully. users can define and describe target platforms using the Hardware Description file
U17	The user shall be able to constrain the target platforms considered by the PHANTOM framework for the deployment of a given parallel code block	SHALL	Fully. users can define in the component network type of target platform for a specific component. More control can be given if user setup a deployment plan
U18	The PHANTOM framework shall hide the target platform heterogeneity, abstracting the underlying platform technologies' details from the user, provided the user is not trying to exploit specific platform capabilities.	SHALL	Fully. PHANTOM API can hide/abstract platform specific configurations from the user
U19	The PHANTOM framework shall be able to generate target dependent parallel code for all mandatory target platforms without user involvement when sufficient annotations are provided.	SHALL	Fully. PHANTOM is able to generate platform-specific code based in the code provided by the user and some Pragmas as annotations

APIs and Annotations			
ID	Description	Priority	Evaluation
U20	The PHANTOM framework shall provide constructs or abstractions to deal with non-uniform and uniform memory, hiding the underlying data transfer details	SHALL	Fully. PHANTOM API provides a shared memory mechanism that abstracts memory access details
U21	The PHANTOM framework shall automate the process of transferring data to/from different memories according to the component data model	SHALL	Fully. PHANTOM API provides a shared memory mechanism that abstracts and simplifies the memory access procedures
U23	The PHANTOM framework shall provide means to indicate data dependencies, defining how data can be partitioned/split among the parallel application components	SHALL	Fully. The PHANTOM Programming Model facilitates the user to define data dependencies in the components maximizing the parallelization capacity of a given application.
U24	The PHANTOM framework shall provide means for the developer to describe the composition of hardware components and interactions for the target platform	SHALL	Fully. PHANTOM framework allows the user to describe hardware composition using the hardware description files
U25	APIs, along with annotations, should be provided to application developers allowing them to statically express which tasks have to be isolated from the others e.g. running on the different CPUs	SHOULD	Fully. users can use the component network description and a set of Pragmas to specify where components should run
U28	PHANTOM shall provide a data model for specification of input and output data	SHALL	Fully. PHANTOM framework allows the user to define data-driven component networks of the user application that specify input and output data flows per component.

Testing			
ID	Description	Priority	Evaluation
U29	PHANTOM should provide a means to test the correct functioning of the software application when it is mapped onto heterogeneous HW targets	SHOULD	Fully. Application functioning is tested via both model validation and functional testing. The mapping between application and HW targets can either be defined by MBT or use MOM decision.
U30	PHANTOM should provide mechanism to test the correct APIs implementation	SHOULD	Fully. The execution of all test cases involves the interaction between MBT and PHANTOM APIs and tests the API implementation.
U31	PHANTOM should provide an API for implementation of tests (similar to JUnit for Java)	SHOULD	Fully. MBT model in xLia and component network schema are APIs to

			achieve early validation; TTCN-3 or MBT models in xLia, used to generate test cases, are APIs to realise test execution.
--	--	--	--

Multi-dimensional optimisation (timing, power, thermal)			
ID	Description	Priority	Evaluation
U49	The mapping proposed by the PHANTOM framework shall reason about the functional and non-functional requirements of the application	SHALL	Fully. MOM has into consideration the both functional and non-functional requirements
U50	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for execution time	SHOULD	Fully. User can specify non-functional requirements using the component network
U51	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for memory	SHOULD	Fully. User can specify non-functional requirements using the component network
U52	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for power consumption	SHOULD	Fully. User can specify non-functional requirements using the component network
U53	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for cost	SHOULD	Fully. Given the cost definition as a combined assessment of execution time and power consumption, MOM generates mappings that may satisfy different cost-effective scenarios.
U54	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for communications bandwidth	SHOULD	Fully. User can specify non-functional requirements using the component network
U55	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for I/O	SHOULD	Fully. The non-functional requirement for I/O usage is consider as a partial analysis within MOM to generate an overall execution time analysis.
U56	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for security	SHOULD	Fully. Requirements regarding security aspects are both explicitly considered in MOM covering execution integrity in GPUs and implicitly covering data security with the use of the Component Network Execution Integrity and the Repository component which integrates data authentication and security related features.

U57	The PHANTOM framework shall consider data obtained from the run-time monitoring of non-functional properties in the mapping step, reconfiguring and optimizing the mapping according with the obtained execution data	SHALL	Fully. MOM and use monitoring data collected on previous execution to re-configure and optimize mappings
U58	The PHANTOM framework should consider data obtained from the run-time monitoring of non-functional properties in the parallelization step, thereby modifying the optimizations to which the code to parallelize will be subjected	SHOULD	Fully. MOM interacts with the PHANTOM monitoring framework to retrieve performance metrics per on non-functional properties (e.g. execution time) per application component to propose the most efficient parallelization granularity.
U60	The PHANTOM framework shall accept qualitative non-functional requirements expressed in the form of an intent to optimize towards a given non-functional property	SHALL	Fully. Qualitative non-functional requirements can be interpreted in to quantitative ones, in the form of bounded values.
U61	The PHANTOM framework shall accept quantitative requirements expressed in the form of bounds, which the non-functional properties should not surpass in run-time	SHALL	Fully. MOM already accepts quantitative requirements of execution time and power consumption in the form of upper bounds.
U62	It should be possible to export the state of the multi-objective mapper and its underlying input data in order to continue development in another workstation / PHANTOM framework instance (e.g. decisions)	SHOULD	Fully. All PHANTOM tools, including MOM, are integrated in a virtual machine toolbox to facilitate overall usage and reusability. MOM as standalone tool can also be used by any third workstation, if configured to use servers where the data of the application is stored.
U63	PHANTOM should support means for expressing task affinities that allow the developer to group processes/threads to run together on specific processors or separately to meet constraints	SHOULD	Fully. The developer is given the option to indicate the grouping of processes/threads to run together on specific processors with the use of a pre-defined mapping that will be enforced for deployment bypassing MOM's analysis.

System and data security			
ID	Description	Priority	Evaluation
U64	Remote target platforms should be able to be secured against eavesdropping through interfaces with external infrastructures for trust/authentication	SHOULD	Fully. Access to remote platforms require prior authentication. TLS can be used to protect data transmissions.
U65	Data obtained through the run-time monitoring should be able to be secured against eavesdropping / unwanted access	SHOULD	Fully. Access to monitoring data on servers required authentication and commu-

			nication can be secured using HTTPS communication
U66	PHANTOM should support means for tasks isolation and information flow control policy	SHOULD	Fully. PHANTOM supports tasks isolation and deployment in more secure target platforms.
U67	PHANTOM should be able to support HPC/Cloud security mechanisms to protect data and control data access	SHOULD	Fully. PHANTOM security mechanism is compatible with HPC/Cloud security mechanisms
U68	PHANTOM shall be able to guarantee data integrity when applications are mapped onto heterogeneous targets	SHALL	Fully. Data integrity is assured by NGAC controls on repository, OS provided isolation of GPU/CPU processes and FPGA tasks isolation

Performance			
ID	Description	Priority	Evaluation
U69	The PHANTOM development framework shall be able to execute on a typical "mid-end" workstation (Core I5, 4GB RAM)	SHALL	Fully. Phantom Reference platform integrates PHANTOM components in a Virtual Machine able to run on a typical workstation
U70	PHANTOM hardware abstraction layer should introduce less than 2 times performance overhead for memory, IO, and other operations	SHOULD	Fully. Tools used during runtime do not introduce overheads that exceed the target.

Dependability			
ID	Description	Priority	Evaluation
U71	PHANTOM may support survivability mechanisms to detect and recover from faults	MAY	Partially. User can detect that a fault occurred on one (or more) PHANTOM tools but PHANTOM cannot recover by itself

Run-time monitoring			
ID	Description	Priority	Evaluation
U72	The PHANTOM run-time monitor shall be able to monitor non-functional properties of an application	SHALL	Fully. Monitoring Framework supports user-specific metrics
U73	The run-time monitor shall be capable of acquiring monitoring data in all mandatory target platforms (e.g. CPU, FPGA, etc) subject to available hardware capabilities	SHALL	Fully. Monitoring Framework support CPU and a defined set of GPUs and FPGAs
U74	The PHANTOM framework should be capable of monitoring execution time properties	SHOULD	Fully. High accuracy on time measurements at the MF-API in order of few ns
U75	The PHANTOM framework should be capable of monitoring memory properties	SHOULD	Fully. Measured the memory on system level as

			total, amount free, used for buffers, used for cache I/O, and amount used. Measured used memory on application level, and at the component levels if implemented as threads or binary executables.
U76	The PHANTOM framework should be capable of monitoring power consumption properties	SHOULD	Fully: a) using the ACME board we measure the consumption on embedded devices . b) on NVidia GPUs when available sensors C) On HPC is calculated based on the CPU load, mem operations, I/O, and network traffic
U77	The PHANTOM framework should be capable of monitoring communications bandwidth properties	SHOULD	Fully: a) Measured the bandwidth used of each network device on system level. b) Measured amount of data sent and received per second on PHANTOM application level, and app component level.
U78	The PHANTOM framework should be capable of monitoring I/O properties	SHOULD	Fully: measure amount of data read and write per on I/O on PHANTOM application level, and app component level.
U79	The data obtained by the run-time monitor shall be accessible and exposed to the user for their own tasks	SHALL	Fully. Data collected can be accesses on the Monitoring server and visualised
U80	The user shall be able to have fine-grained control over execution time monitoring by indicating application sub-components (i.e. tasks, loop, code blocks) whose execution time shall be monitored	SHALL	Fully. Using User defined metrics can monitor the execution time of any size of code tasks, looks, code blocks), accuracy in order of few ns.
U81	It should be possible to export the run-time monitoring data in a structured data format	SHOULD	Fully. Data can be exported in JSON format
U82	For non periodic non-functional properties, the user should be able to select the frequency of data acquisition	SHOULD	Fully. The configuration parameters allow defining the frequency of data acquisition independently for each plugin and each application.
U83	PHANTOM should provide monitoring of application-specific performance metrics	SHOULD	Fully. Monitoring Framework supports user-specific metrics
U84	PHANTOM shall provide a facility for storing and retrieving historical profiles of the stored events and metric values	SHALL	Fully. Data collected can be accesses on the Monitoring server and visualised

U85	PHANTOM shall provide the possibility to perform some basic analytics for the stored performance data	SHALL	Fully. PHANTOM Monitoring server calculates average maximum and minimum of data stored. The RESTful API allows users to submit ElasticSearch scripts.
-----	---	-------	--

Communications			
ID	Description	Priority	Evaluation
U86	PHANTOM may support application specific communication bus/protocols	MAY	Partially. Components have been implemented in a network protocol independent way. While PHANTOM does not provide specific mechanisms to support communication bus/protocols like serial communication, it also does not block implementations on applications

5. CONCLUSIONS

In this document was presented the Reference Integrated System that integrates the PHANTOM tools in two virtual machines. Those machines are especially useful for the to allow new users to test and get used to PHANTOM technologies and its functionalities before advancing to customized, potential more efficient, tool setups. This virtual environment is complemented with by the description of the detailed tool flow between the tools implemented in the Reference Integrated System. Also, and to further ease the effort required to new PHANTOM Users, it is provided a set of User-Scripts that allow to manage some elements of the virtual environment, like the local deployed server, and to simplify the upload and analysis and execution of the User application.

As one of the main elements that support the integration between PHANTOM tools, it is also presented the Repository, with its REST API, the context description supported by a dynamic metadata description and its Subscription/Notification mechanism, that explores the advantages of the metadata description mechanism to support the expression to queries to different types of contents or events. The Repository also acts as a security element by facilitating the authentication mechanism and by continually assessing the rights of the User using the security framework.

Finally, the requirements related with the overall of the PHANTOM framework, its capabilities and design, are assessed by the R&D partners. From a total of 55 requirements marked as ‘Shall’, ‘Should’ and ‘May’ (representing priority from the greater to the lesser), only two requirements were evaluated as ‘Partially’ met, and whose priority as classified as ‘May’. All the other requirements were evaluated as ‘Fully’ met.