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

Project Number 688146

D2.2 – Final report on system software for multi-dimensional optimization on heterogeneous systems

Version 1.0
31 August 2018
Final

Public Distribution

WINGS ICT Solutions, University of York, The Open Group, University of Stuttgart

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

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 Tel: +39 050 9657 513 E-mail: silvia.mazzini@intecs.it	**The Open Group** Scott Hansen Rond Point Schuman 6 5th 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
0.1	Internal draft	16/07/2018
0.4	Partners' contributions	31/07/2018
0.5	Version for QA	1/08/2018
0.8	Revised version by contributors	9/08/2018
0.9	Version for internal review	17/08/2018
1.0	Final version	31/08/2018

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This document describes the final release of the system software stack for multi-dimensional optimization of the PHANTOM platform, as resulted from the actions performed by the workpackage WP2. The goal of this software stack is to enable applications that are developed with the PHANTOM platform to be executed on resources of a heterogeneous infrastructure environment, in a way that helps them comply with the user's functional and non-functional requirements, identified previously. With this goal in mind, the PHANTOM platform provides the following tools:

- **The PHANTOM Generic Multi-Objective Mapper –** a software framework which analyses the historical data that characterise the previous executions of the application in order to propose their optimal dynamic mapping on the heterogeneous hardware platform. The generic MOM strives to optimize the application characteristics according to the actual (at the time of application deployment) utilization of the hardware resources and finds the deployment configuration that complies most with the user's requirements. The deployment configurations that are generated by MOM are thus dynamic and can change for every new deployment request for a particular application.

  The developed version of the Generic MOM full prototype is covering non-functional requirements focusing on time performance (both computation and communication), power consumption, as well as memory utilization. Data movement is considered by the minimization of the communication time. Functional requirements are fulfilled by the execution of the application as defined by the user. All three use cases are currently supported. The current version of the Generic MOM supports in full extent CPU-GPU based architectures and CPU-FPGA based architectures. This full version integrates security, by taking as input the component network as derived by the application of the Component Network Execution Integrity.

- **The PHANTOM Offline Multi-Objective Mapper –** an extension of the Generic MOM which is able to analyse the properties of newly-submitted applications, for which the dynamic mapping might be not efficient. The Offline MOM tests proposed deployments and rejects those that will fail to meet the user's requirements. The technique is particularly important for the applications that expose real-time requirements (like the Telecom and Surveillance use cases).

  The primary innovation in this approach is the seamless integration of the state-of-the-art into the development process in a way that allows the value of that research to be used by non-experts. In addition, the Component-based Programming Model makes possible to analyse the system in parts. This is an advance over traditional real-time analysis, which attempts to consider the entire system as a whole.

- **The PHANTOM Monitoring Server and the Execution Manager –** a software stack for storing and analysis of the application- and hardware-specific data (in form of special metrics). The Monitoring Server and the Execution Manager provide the functionalities to query and analyse the registered data.

The Execution Manager is used to initiate tracking of all running PHANTOM applications.

The developed Interface allows to the users and PHANTOM tools to analyse the collected monitored metrics across multiple executions of the applications. The Interface is extended also for the user-defined metrics, which can give important hints for PHANTOM tools like MBT.
The Monitoring Server and Execution Manager are integrated with Grafana and Kibana visualization back-ends, and also provide RESTful API that enables the visualization through the command line (in a textual form).

- **The PHANTOM Security Framework –** an environment that aims to enforce the security policies of the PHANTOM applications in heterogeneous and distributed hardware infrastructure.

  The framework supports execution integrity for heterogeneous computing devices and NGAC-based access control integrated with the PHANTOM Repository and Execution Manager, based on a lightweight 'ngac' server.

All tools have been subject to optimization and improvement of the technologies that were introduced in Deliverable D2.1. All improvements have been guided by the preliminary feedback from the use case providers, done during the M18 pilot evaluation. This document includes details on the final design, technical decisions and know-how, innovations and usage perspectives for each of the targeted tools.

The remainder of the document is organized as follows. Section 1 discusses rationale and motivation for the developed tools. Sections 2-5 discuss the tools in detail, including the imposed user requirements, implementation and integration within the PHANTOM platform. Section 6 highlights the major achievements and discusses impact on the potential user communities.

1. INTRODUCTION

1.1 RATIONALE AND MOTIVATION

Multi-dimensional optimization is one of the key technologies in high-performance computing, aiming to efficiently cope with the trade-offs between an ever-increasing demand for higher performance in terms of processing time and energy consumption of multi-core processing platforms. A great challenge is the exploration of the available hardware infrastructure beyond homogeneous multi-core platforms, to heterogeneous platforms including CPU/GPU, CPU/FPGA and embedded/cloud platforms. PHANTOM multi-dimensional optimization technologies aim to efficiently manage the mapping of the application components to heterogeneous platforms, providing dynamic application code migration towards optimized energy efficiency, higher performance, while also considering security aspects.

Current approaches in multi-dimensional optimization consider conventional optimization techniques for a set of objectives, which in most cases are limited to execution time and energy consumption. In this direction, the FiPS [1] project focuses on reducing the power performance ratio within data-centres by improving the usability and usefulness of heterogeneous platforms that incorporate FPGAs, embedded CPUs, GPUs and multicore accelerators into high-performance and low-power heterogeneous computing servers. Furthermore, the ALMA project [2] aimed at developing a tool-chain that enables the efficient mapping of applications on multiprocessor platforms from a high level of abstraction, focusing on reduced development cost and performance. A dynamic approach was proposed by the DreamCloud project [3], which provides dynamic resource allocation in many-core embedded and high-performance systems, focusing on appropriate guarantees on performance and energy efficiency.

In addition to execution time and power consumption, *Multi-Objective Optimization* technologies in the context of PHANTOM consider additional non-functional properties such as memory utilization, data security, and temperature. Furthermore, the optimization mechanisms will consider resources and requirements from heterogeneous platforms (e.g. GPU, FPGA) in a unified manner, splitting an application and mapping the application's execution on different technologies (e.g. CPU-GPU, CPU-FPGA) at the same time, rather than mapping the application on one platform at a time. The optimization process is assisted by Model-Based Testing mechanisms, which evaluate Quality of Service at the functional level regarding the application execution and also provide an estimation of the application's requirements satisfaction when not provided by the monitoring mechanisms.

The optimization process in most of the state-of-the-art optimization tools is restricted to non-functional requirements set by the developers, or hardware restrictions derived from the available hardware resources. Some tools also provide the user with the ability to monitor the desired requirements, providing additional knowledge for further manual optimization. The multi-objective optimization in the context of PHANTOM is performed automatically, enhanced by the *Monitoring Framework* that is able to monitor several types of resources (e.g. computation, I/O, storage, and network traffic) in a unified manner, offering on-the-fly data analysis, prediction and visualization

capabilities that will further interact with the mapping mechanisms, by providing them with more accurate feedback for optimization actions.

In addition, *security* is another major aspect of both the application and the underlying software/hardware, when moving applications and portions of data among heterogeneous platforms, especially in Cloud systems. Security integration is a challenging task that may introduce performance degradation. The majority of optimization frameworks address security in terms of management of conflicts (data and access) in a proactive manner, rather than implementing actual access control procedures. The PHANTOM framework considers security in terms of data and execution integrity using resource isolation and access controls. Furthermore, the PHANTOM security framework supports fine-grained access control by an application's subcomponents, according to security policies, established automatically by the PHANTOM framework or manually by the developer, complementing the optimization process with application robustness.

1.2 SCOPE OF ADDRESSED TOOLS

This deliverable addresses the tools that implement the integrated cross-layer, multi-objective and cross-application approach in PHANTOM platform. In particular, the following 4 tools are in the scope of the deliverable:

- *Scheduling solutions*

  (1) Generic Multi-Objective Mapper

  Performs adaptive mapping of complex application workflows, leveraging the actual resource consumption and past application execution statistics

  (2) Offline Multi-Objective Mapper

  Analyses mapping offline to reject those that can be shown to fail to the user's requirements.

- *Monitoring solutions*

  (3) Monitoring Server and Execution Management Service

  Collects statistics of applications and hardware resources and provides a rich analysis functionality for them.

- *Security enforcement solutions*

  (4) Security Framework

  Provides tools for controlling and enforcing resource isolation and access control across all levels starting from isolated protection domains supported by hardware and software.

Figure 1 shows a representation of the PHANTOM architecture, where are higlighted the components addressed in WP2.

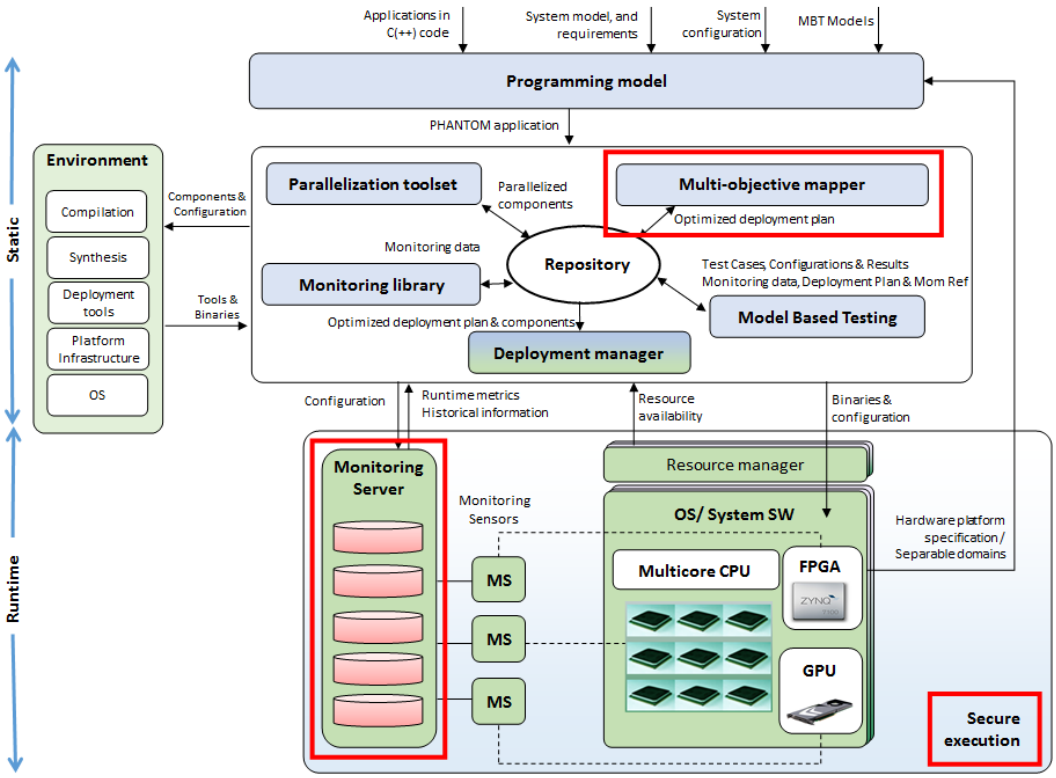


Figure 1. Components of the PHANTOM architecture addressed in WP2

The description of the tools in this deliverable capitalizes on information and details that are further described in existing PHANTOM deliverables, as described in the following paragraphs:

- Use case requirements, with a potential link with D1.1.
- Design specifications, with a potential link with D1.2.
- Implementation details, with a potential link with D4.1.
- Testing results (if available), with a potential link with D5.1.
- Dependencies/integration aspects, with a potential link with D1.2.

In addition, technologies/components developed in WP2 need to interact under the tool flow, specified in D1.2, with other components of the PHANTOM platform, which are the subject of D3.1 (i.e. Parallelization Toolset and Model-Based Testing) and D4.2 (Heterogeneous HW platforms). Information about these dependent technologies can be found in the corresponding deliverables. The multi-dimensional optimization process is driven by the multi-objective mapping technologies supported by extensive runtime monitoring functionality and security mechanisms.

The *Multi-Objective Mapping* technologies consist of a generic mapper and an offline mapper, interacting among them. The generic mapper explores the potential mappings of components to the available infrastructure resources to choose the optimal one

according to non-functional metrics. The offline mapper then performs a more focused search towards strict timing closure constraints in order to further optimize the deployment towards execution time and to ensure the timing constraints.

The multi-dimensional optimization process is supported by a *runtime Monitoring Framework* that is able to perform several types of runtime monitoring of both application and system resources, and to store the results to a monitoring server. The monitoring framework collects a diverse set of data and translates them to more specific metrics, which are forwarded to the Multi-Objective Mapper modules to assist the optimization process towards more accurate decisions. The monitored metrics are also processed by the data analytics tools that enhance the optimization process with diagnostics, and enable visualization by means of data correlation, and prediction of upcoming states and events.

In order to ensure undisrupted operation and avoidance of unauthorized access, PHANTOM incorporates *security mechanisms* focused execution integrity through data isolation and information flow control at both software and infrastructure levels, along with policy-based access control overseen by a policy server functions running on one of the computational nodes.

This deliverable describes the appropriate methods and APIs for the integration of the aforementioned technologies into a holistic multi-dimensional optimization framework and the overall PHANTOM platform. In addition to this description, some details of the integration of the tools will be described at WP5 "*Integration, use case implementation and validation*".

1.3 AMBITIONS AND MAJOR INNOVATIONS

The purpose of these mechanisms is to collaborate to provide an optimized mapping of application components to heterogeneous platforms. In particular, the optimized mapping is obtained based on a multi-dimensional optimization process considering non-functional requirements, with enhanced accuracy and cognition provided by the monitoring framework, along with enhanced robustness provided by PHANTOM security mechanisms.

Next, the major innovations of these mechanisms are summarized:

- The Generic MOM innovation relies on the following features:

Multi-dimensional optimization targeting systems with increased heterogeneity (CPU, CPU-GPU, CPU-FPGA) across the computing continuum (from embedded to compute clusters).

The Multi-dimensional optimization in these systems is assisted by Model-Based Testing for estimation and verification of application's requirements satisfaction and Monitoring Framework to monitor both resources and application.

- The Offline MOM innovation relies on the following features:

The seamless integration of the state-of-the-art into the development process in a way that allows the value of that research to be used by non-experts. The platform automatically makes use of the Offline MOM to reject failing mappings early, thereby saving development time.

The Component-based Programming Model was defined in order to assist with real-time analysis. By requiring applications to be componentised and their communications explicitly enumerated, it is possible to analyse the system in parts. This is an advance over traditional real-time analysis that attempts to consider the entire system as a whole. This still relies on a communications platform/protocol that has determinable worst-case performance guarantees.

- Monitoring Framework Server + Execution Manager innovation relies on the following features:

The monitoring server improves the efficiency by allowing the data analytics to be performed where the data is stored. It is more efficient than sending a large amount of data to be processed in another location.

The separation of gathering, storing and analysis technologies. The MF-Server, playing the role of the interface, can provide a consistency filter of the collected data, and filter possible errors. The analysis of previous executions and generation of statistics is done by the Execution Manager which can prepare the request of such results as soon as the metrics are available, improving the response time.

The analysis functionality can be easily extended to the users' needs. Both MF-Server and Execution Manager are open-source and both support user-defined micro-queries directly submitted by the users.

The users can access all the stored data in the database, which can support to debug their tools and analyse the behaviour of the user applications.

Highly scalable on distributed platforms: MF-server and the Execution Manager, as well as the ElasticSearch database, can run on distributed platforms.

- Security innovation relies on the following features:

Innovations include the refinement of the NGAC functional architecture to place the resource access path into the hands of the application developer, a lightweight NGAC Policy Server with RESTful Policy Query Interface API, and a declarative policy language accepted by the NGAC policy tool. This all makes access control a first-class part of the development model and automatically enforceable by the platform.

2. THE PHANTOM GENERIC MOM

The Multi-Objective Mapper (MOM) is responsible for optimising the mapping of components and shared data communications throughout the target architecture, towards user-defined non-functional requirements, while maintaining Quality of Service. MOM considers evolutionary/bio-inspired multi-objective optimization approaches in order to optimize the placement of components against multiple objectives such as power, performance (execution time, energy consumption, memory utilization but also data security), considering requirements/policies from application developers' side, as well as the status and capabilities of computing resources. The MOM primarily operates offline to decide the static mapping of the source application components, considering information from the Parallelization toolset, the Model-Based Testing, and requirements from the PHANTOM Security mechanisms. However, MOM may also operate online for dynamic mapping optimization, interacting with PHANTOM Monitoring mechanisms in order to achieve optimized mapping according to the most recent monitored data.

2.1 REQUIREMENTS

2.1.1 Use case requirements

The use case requirements related to the Multi-Objective Mapper, are mainly referring to performance in terms of execution time, energy consumption, memory utilization but also data security and other use case requirements, as defined in D1.1. The addressed requirements are of equal significance to all three PHANTOM use cases: Surveillance, Telecommunications and High-Performance Computing. Indicatively, we list below the most representative requirements that are related to Generic MOM referring to all use cases:

- Mapping of parallel code blocks to target platforms;
- The user shall be able to enforce a given mapping, overriding the proposed one depending on existing constraints and availabilities;
- Qualitative non-functional requirements expressed in the form of an intent to optimize towards a given non-functional property.

The full list of use case requirements related to Generic MOM is provided from the following numbers which correspond to D1.1: U48, U49, U50 to U56, U57, U59, U60, U61, and U63. Based on the current status of MOM, all below requirements can be considered satisfied, except for U60 that will be finalized as part of the integration activities in WP5.

Table 1. Use case requirements addressed by Generic MOM

Req. No.	Requirement	Priority	Status
U48	The PHANTOM framework	SHALL	Achieved. The Generic MOM gen-

Req. No.	Requirement	Priority	Status
	shall propose a mapping of the parallel code blocks, composing the application, to the available resources/target platforms		erates a deployment plan in the form of an XML specification to define mappings of the parallel application code block to the available resources on the target hardware platforms
U49	The mapping proposed by the PHANTOM framework **shall** reason about the functional and non-functional requirements of the application	SHALL	Achieved. The mappings generated by MOM fulfill by default all the functional requirements of the application. The same applies for the non-functional requirements (time, power consumption). However, different mappings satisfy the requirements in distinct degrees.
U50	The PHANTOM framework **should** provide facilities (e.g. APIs or annotations) to consider non-functional requirements for execution time	SHOULD	Achieved. The mappings generated by MOM fulfill by default all the non-functional requirements of the application including execution time. MOM interacts with the Monitoring Framework which provides facilities such as specific monitoring API that is used as part of the Deployment Manager activities to generated application binaries for execution and support execution time monitoring.
U51	The PHANTOM framework **should** provide facilities (e.g. APIs or annotations) to consider non-functional requirements for memory	SHOULD	Achieved. The mappings generated by MOM fulfill by default all the non-functional requirements of the application including those set for targeted memory usage.
U52	The PHANTOM framework **should** provide facilities (e.g. APIs or annotations) to consider non-functional requirements for power consumption	SHOULD	Achieved. The mappings generated by MOM fulfill by default all the non-functional requirements of the application including power consumption. MOM interacts with the Monitoring Framework provides facilities such as specific monitoring API that is used as part of the Deployment Manager activities to generated application binaries for execution and support power consumption monitoring.
U53	The PHANTOM framework **should** provide facilities	SHOULD	Achieved. Given the cost definition as a combined assessment of execu-

Req. No.	Requirement	Priority	Status
	(e.g. APIs or annotations) to consider non-functional requirements for cost		tion time and power consumption, MOM generated 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	Achieved. The non-functional requirements for communications bandwidth are consider as a partial analysis within MOM to generate an overall execution time analysis.
U55	The PHANTOM framework **should** provide facilities (e.g. APIs or annotations) to consider non-functional requirements for I/O	SHOULD	Achieved. 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	Achieved. 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 (see 5.3.1) 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	Achieved. This mechanism is already in place and is used to drive MOM's analysis when generating mappings that target specific user performance requirements.
U59	The user **shall** be able to enforce a given mapping, overriding the proposed one depending on existing constraints and availabilities	SHALL	Achieved. The enforcement of a user define mapping is already in place by MOM. Inputs mapping(s) are set as configuration arguments upon MOM's execution and may bypass MOM's analysis on demand.
U60	The PHANTOM framework **shall** accept qualitative non-functional requirements expressed in the form of an	SHALL	Achieved. Qualitative non-functional requirements will be interpreted in to quantitative ones, in the form of bounded values. This

Req. No.	Requirement	Priority	Status
	intent to optimize towards a given non-functional property		feature will considered as we finalize the integration activities in MOM.
U61	The PHANTOM framework **shall** accept quantitative requirements expressed in the form of bounds, which the non-functional properties should not surpass in runtime	SHALL	Achieved. MOM already accepts quantitative requirements of execution time and power consumption in the form of upper bounds.
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	Achieved. The developer is given the option to indicate the grouping of processes/threads to run together on specific processors with the use of a predefined mapping that will be enforced for deployment bypassing MOM's analysis.

2.1.2 Preliminary use case feedback from M18 results

The Multi-Objective Mapper is able to provide mappings for the use cases on the available hardware platforms. Furthermore, the tool was able to mutate the mappings, producing better ones, using mutations related to both communication objects' remapping, based on the memory access time, and further parallelization of the component with the highest execution time.

The tool is able to evaluate the produced mappings according to the requirements provided for each use case and the metrics obtained by MBT and the Monitoring Framework. The mutation loop is executed to populate the Pareto optimal [4] solutions, until the user defined number of iterations is reached or until the mutation process cannot provide better mapping for several executions. The homogeneous and scalable configurations of CPU-based hardware have been evaluated. In addition, the heterogeneity of the approach has also been validated with the integration of GPU and FPGA-based platforms as well. Furthermore, the second version of MOM prototype integrated power and security requirements in the mapping evaluation process.

MOM can be invoked in a command prompt using an easy-to-use parameterized command line and the time-performance of the tool was acceptable.

2.2 DESIGN SPECIFICATIONS

The goal of Generic MOM is to optimise the deployment of the user application components to the underlying hardware platforms for user requirements satisfaction and maximum performance in terms of execution time, energy efficiency, security and other use case requirements, as defined in D1.1. To that end, MOM needs to interact with

other components in PHANTOM architecture (as shown in Figure 2 and Figure 3 below), namely the Model-Based Testing, the Parallelization toolset, the Monitoring library and framework, as well as the Repository, from which MOM receives its input and stores its outcome.

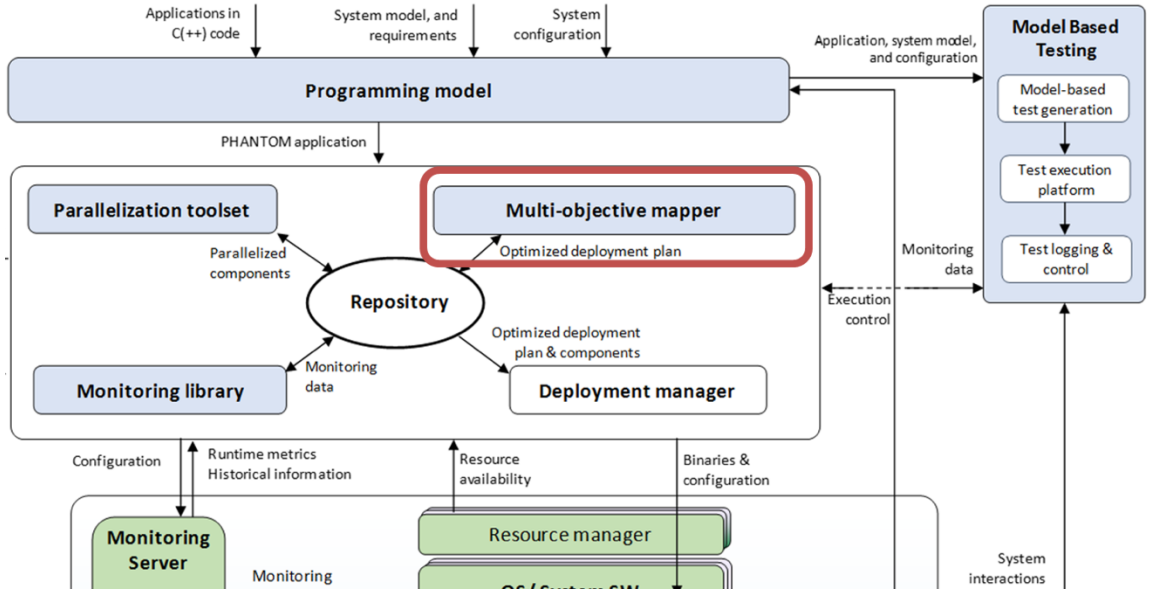


Figure 2: **MOM positioning in PHANTOM architecture**

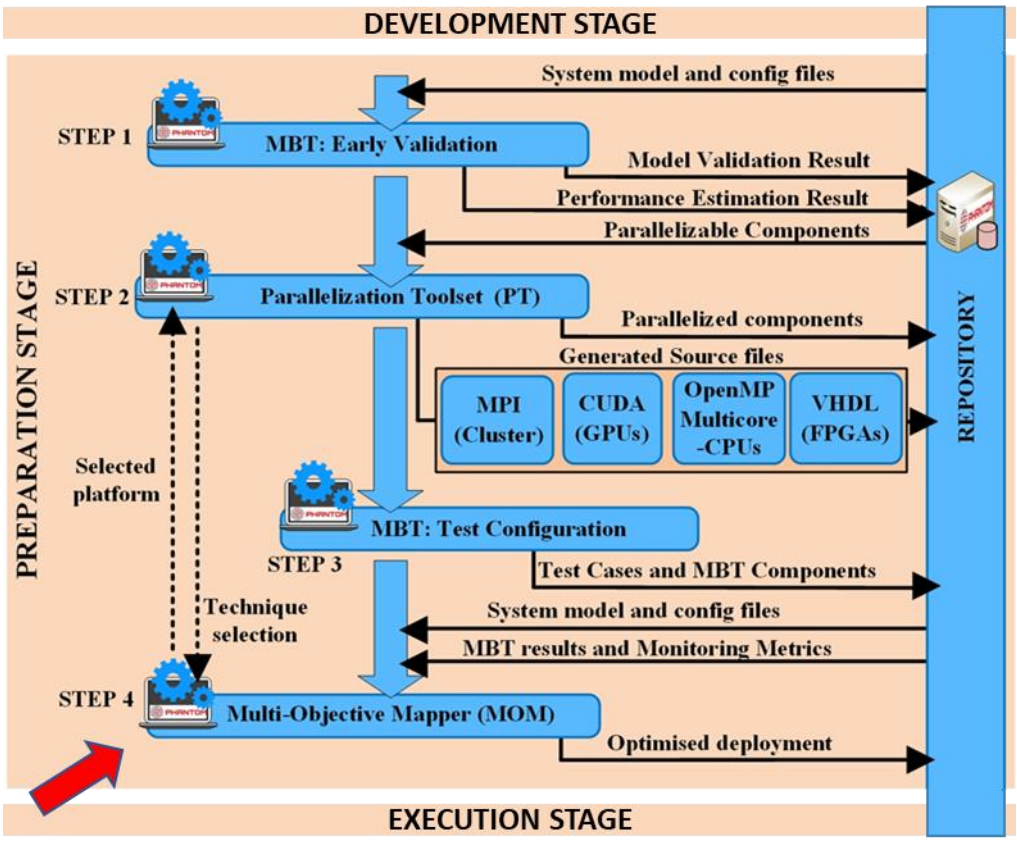


Figure 3: MOM positioning in PHANTOM tool-flow

In this direction, MOM requires the definition of the hardware infrastructure and the application's description in a PHANTOM specific format. Specifically, MOM requires as input the System Model (component network specification), the Platform Model (hardware platform specification) and the System Configuration (user defined requirements and, if available, the initial mapping). Furthermore, MOM collects estimations on the non-functional requirements from the Model-Based Testing toolset, while results from previous application deployments on the underlying hardware platform (Performance data) are derived from run-time monitoring through the Monitoring Library. In addition, the Parallelization toolset provides MOM with parallelization directives to further assist MOM in the mapping optimization process. The Multi-Objective Mapper is then performing a number of optimization steps, executing a multi-objective optimization evolutionary/bio-inspired algorithm, which considers all the above input specification and the related requirements. Finally, MOM generates the Optimised deployment plan (mapping decision of application components to hardware platform processing elements and of communication objects to hardware platform physical memories). All interactions that MOM performs with the rest of the PHANTOM components are facilitated and implemented through the use of the Repository. The latter provides secure interactions using a token mechanism to manage the access rights of PHANTOM users.

2.2.1 Problem formulation

Initially the Generic MOM receives the System Model including the component network specification, the Platform Model including the hardware platform specification, and the System Configuration including an initial mapping.

The component network specifies a set of application components, communication objects, and relations between communication objects and components defining the source and destination components.

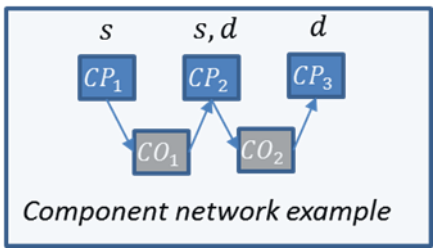


Figure 4: Component network example

Assuming a set of application components:

$$\boldsymbol{CP} = \ < \boldsymbol{CP_1}, \ \ldots, \boldsymbol{CP_N} > \tag{1}$$

where N is the number of components, and a set of communication objects:

$$\boldsymbol{CO} = \ < \boldsymbol{CO_1}, \ \ldots, \boldsymbol{CO_M} > \tag{2}$$

where M is the number of communication objects.

We define a relation:

$$\boldsymbol{P}: \boldsymbol{CO} \rightarrow \boldsymbol{CP} \tag{3}$$

s.t. we have a subset

$$\boldsymbol{P} \subset \boldsymbol{CP} \times \boldsymbol{CO}, s.t. \forall \boldsymbol{co} \in \boldsymbol{CO}, \exists \boldsymbol{s}, \boldsymbol{d} \in \boldsymbol{CP}, \boldsymbol{s.t.}(\boldsymbol{co}, \boldsymbol{s}, \boldsymbol{d}) \in \boldsymbol{P} \tag{4}$$

where s is the source and d is the destination component.

The second input of MOM is the hardware platform which is defined as a list of processing elements (C CPUs, F FPGAs, G GPUs, V Other specialised acceleration devices):

$$\boldsymbol{PE} = \ < \boldsymbol{PE_1}, \ \ldots, \ \boldsymbol{PE_C}, \boldsymbol{PE_{C+1}}, \ \ldots, \boldsymbol{PE_{C+F}}, \boldsymbol{PE_{C+F+1}}, \ \ldots, \ \boldsymbol{PE_{C+F+G}}, \boldsymbol{PE_{C+F+G+1}}, \boldsymbol{PE_{C+F+G+V}} > \tag{5}$$

connected to physical memories of different type (CM CPU memories, FM FPGA memories, GM GPU memories, VM Movidius memories) :

$$\boldsymbol{MEM} =< \boldsymbol{MEM_1}, \ \ldots, \ \boldsymbol{MEM_{CM}}, \boldsymbol{MEM_{CM+1}}, \ \ldots, \boldsymbol{MEM_{CM+FM+GM+VM}} > \tag{6}$$

via connections defined as relations R.

The PHANTOM heterogeneous parallel infrastructure, comprising all the above defined processing elements, is described in detail in D4.2 section 2.

$\forall\ mem \in MEM$ we define a relation

$$\boldsymbol{R}: \boldsymbol{PE} \rightarrow \boldsymbol{MEM}, \tag{7}$$

where $\forall mem \in MEM$, $\exists$ a subset

$$S \in PE,\ s.t.\ (mem,\ S) \in R \tag{8}$$

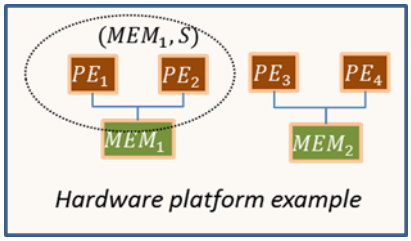


Figure 5: Example of hardware platform

In addition, MOM receives an initial mapping in order to start the exploration and generation of the optimized mappings. Each mapping consists of mappings of components to processing elements and mappings of communication objects to physical memories.

A component mapping to processing elements is defined as:

$$\boldsymbol{Map^c}: \boldsymbol{CP} \rightarrow \boldsymbol{PE}, \tag{9}$$

where we define a subset $Map^c \subset CP \times PE$, such that:

$$\forall\ \boldsymbol{cp} \in \boldsymbol{CP},\ \exists\ \boldsymbol{pe} \in \boldsymbol{PE},\ \boldsymbol{s.t.}\ (\boldsymbol{cp},\ \boldsymbol{pe}) \in \boldsymbol{Map^c}, \tag{10}$$

and

$$\forall\ (\boldsymbol{cp},\ \boldsymbol{pe}) \in \boldsymbol{Map^c}\ \boldsymbol{and}\ (\boldsymbol{cp},\ \boldsymbol{pe'}) \in \boldsymbol{Map^c}\ \Longrightarrow \boldsymbol{pe} = \boldsymbol{pe'} \tag{11}$$

A communication objects mapping to memories is defined as:

$$\boldsymbol{Map^m}: \boldsymbol{CO} \rightarrow \boldsymbol{MEM}, \tag{12}$$

where we define a subset $Map^m \subset CO \times MEM$, such that:

$$\forall\, \boldsymbol{co} \in \boldsymbol{CO},\ \exists\, \boldsymbol{mem} \in \boldsymbol{MEM},\ \boldsymbol{s.t.}\ (\boldsymbol{co},\ \boldsymbol{mem}) \in \boldsymbol{Map^m} \tag{13}$$

and

$$\forall\, (\boldsymbol{co},\ \boldsymbol{mem}) \in \boldsymbol{Map^m}\ \boldsymbol{and}\ (\boldsymbol{co},\ \boldsymbol{mem'}) \in \boldsymbol{Map^m}\ \Longrightarrow \boldsymbol{mem} = \boldsymbol{mem'} \tag{14}$$

According to the above a Mapping is defined as:

$$\boldsymbol{Map} = \boldsymbol{Map^c} \cup \boldsymbol{Map^m} \tag{15}$$

The following figure provides an example of mapping:

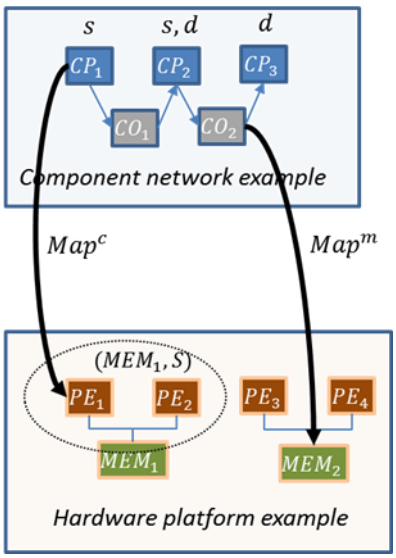


Figure 6: Example of mapping

The Component network, Platform description, and initial deployment plan restrict MOM's exploration possibilities with a set of constraints, as defined in the following description. The first type of mapping constraints are derived from the component network, where, based on a mapping $m \in Map$, and considering cp_1 and cp_2, with $cp_1 \xrightarrow{m} pe$ and $\forall\, co \in CO'$: $(co, cp_1, cp_2) \in P$,

if

$$\boldsymbol{co} \xrightarrow{\boldsymbol{m}} \boldsymbol{mem} \text{ and } \boldsymbol{pe'} \notin \boldsymbol{S}:\ \boldsymbol{R}(\boldsymbol{mem}, \boldsymbol{S}), \tag{16}$$

then

$$\boldsymbol{cp_2} \notin \boldsymbol{CP'}: \boldsymbol{CP'} \overset{\boldsymbol{m}}{\Rightarrow} \boldsymbol{pe'} \tag{17}$$

In addition, based on a mapping $\in Map$,

$\forall\, co \in CO'$: $(co, cp_1, cp_2) \in P,\ with\ cp_1, cp_2 \in CP \xRightarrow{Map^c} < pe_1,\ pe_2 >$.

Let $mem \in MEM$,

if

$$< pe_1, pe_2 > \notin R(mem, S), \tag{18}$$

Then

$$co \notin CO' : CO' \overset{m}{\rightarrow} mem \tag{19}$$

Further constraints are considered regarding maximum memory size and defined as follows:

Let

$$MMZ = < MMZ_1, \ldots, MMZ_{CM+FM+GM+VM} > \tag{20}$$

the vector of maximum memory sizes and

$$OMZ(m) = < OMZ_1(m), \ldots, OMZ_{CM+FM+GM+VM}(m) > \tag{21}$$

the occupied size of memories based on a given mapping $m \in Map$, then:

$$OMZ(m) \leq MMZ \tag{22}$$

In addition, PHANTOM supports user-defined requirements described such as the Total execution time, where given a mapping m:

$$TCET(m) \leq TCET_{max} \tag{23}$$

and the Total energy consumption, given a mapping m:

$$EC(m) \leq EC_{max} \tag{24}$$

Same constraints can be defined per component

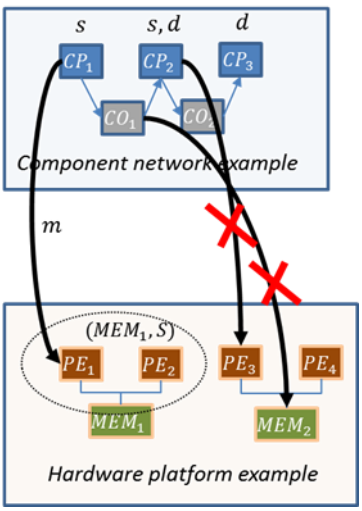


Figure 7: Example of constraint mapping operation

As depicted in the above picture, due to execution time and memory size constraints the component CP_2 is not allowed to be mapped on processing element PE_3 and communication object CO_1 is not allowed to be mapped on memory MEM_2.

After the initial mapping and constraints specification, MOM measures the values of the specified non-functional requirements for each possible mapping, in the form of objective functions. The set of objective functions is represented in the form of vector:

$$\boldsymbol{f(x)} = \ < \boldsymbol{f_1(x)}, \ldots, \boldsymbol{f_L(x)} > \tag{25}$$

Where each objective function f_l, $l \in [1, L]$, represents a corresponding non-functional requirement to be fulfilled (total execution time, energy, etc.).

The following equations provide the representation of the execution time requirement as an objective function.

At first the values of estimated execution time per component and per mapping are defined based on static analysis, MBT result or post-execution monitoring results

$$\boldsymbol{EM(m)} = \begin{Bmatrix} \overrightarrow{\boldsymbol{ETC_1}} & \ldots & \overrightarrow{\boldsymbol{ETC_C}} \\ \ldots & \ldots & \ldots \\ \overrightarrow{\boldsymbol{ETV_1}} & \ldots & \overrightarrow{\boldsymbol{ETV_V}} \end{Bmatrix}, \tag{26}$$

where:

$$\overrightarrow{\boldsymbol{ETC_i}} =< \boldsymbol{ETC_i^1}, \ \ldots, \boldsymbol{ETC_i^{CM}} >, \boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{C}], \tag{27}$$

$$\overrightarrow{\boldsymbol{ETF_i}} =< \boldsymbol{ETF_i^1}, \ \ldots, \boldsymbol{ETF_i^{FM}} >, \boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{F}], \tag{28}$$

$$\overrightarrow{\boldsymbol{ETG_i}} =< \boldsymbol{ETG_i^1}, \ \ldots, \boldsymbol{ETG_i^{GM}} >, \boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{G}], \tag{29}$$

$$\overrightarrow{\boldsymbol{ETV_i}} =< \boldsymbol{ETV_i^1}, \ \ldots, \boldsymbol{ETV_i^{VM}} >, \boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{V}] \tag{30}$$

Then the execution time per application component can be represented as:

$$\boldsymbol{CET(m)} = \ < \boldsymbol{CET_1}, \ \ldots, \boldsymbol{CET_N} > \tag{31}$$

In case all components are mapped on parallel processing elements, the total execution time is provided by:

$$\boldsymbol{TCET} = \mathbf{max}\{\overrightarrow{\boldsymbol{CET(m)}}\}, \tag{32}$$

or in case all components are mapped on the same processing element, then the total execution time is derived by the following equation

$$\boldsymbol{TCET} = \boldsymbol{sum}\{\overrightarrow{\boldsymbol{CET(m)}}\}, \tag{33}$$

(The above statement does not cover all possible cases that will be further investigated)

The corresponding objective function is represented as

$$\boldsymbol{f_1(x)} \ = \ \boldsymbol{TCET} \tag{34}$$

These formulations can be used for the rest objective functions, addressed by generic MOM.

2.3 IMPLEMENTATION DETAILS

The optimized mapping is generated from an evolutionary/bio-inspired multi-objective algorithm, inspired by weighted and genetic algorithms, which is optimized towards PHANTOM requirements. The generic MOM's evolutionary multi-objective algorithm is a custom designed genetic algorithm implemented in JAVA, using appropriate XML libraries to be able to parse its input and produce its output. The generic MOM receives and processes the component network specification along with the hardware platform specification and also an initial mapping, in order to produce the optimized deployment plan, as described in the following steps:

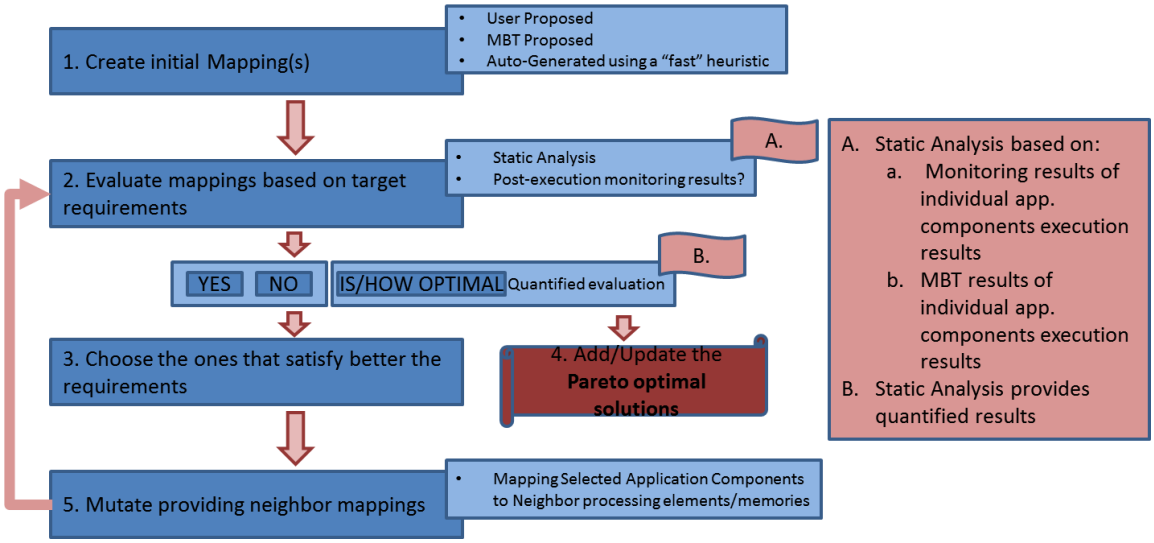


Figure 8: High-level representation of Generic MOM algorithm

Step 1: At the first step, initial mappings are created using a random mapping function. The number of initial mappings is user defined (passed as argument). User may define a custom initial mapping. The satisfaction of all supported requirements is checked upon mapping creation. If a mapping violates the requirements, it is discarded and another one is created to replace it. This applies to the mutated mappings as well. The current supported requirements are user defined and range from total execution time, power consumption, component network connections, underlying hardware characteristics (HW architecture and connections), to memory size (data accommodation capacity).

Step 2: After each mapping is created, an evaluation function is called to evaluate the generated mappings performance based on the given user-defined requirements (time, power, etc.). MOM tool proceeds to a quantified estimation of parameters such as the computation times, the total execution time and the power consumption, based on the metrics (CPU speed, memory size and memory access time) provided by the Platform Model (hardware platform specification) and the Monitoring Framework.

Step 3: The best mapping is selected based on the performance on the given user-defined requirements and is added to the Pareto optimal solutions list. Currently, it is based on the performance towards both the application's total execution time and power consumption.

Step 4: The next step consists of the Mutation of the best mapping providing neighbour mappings, according to the user defined requirements in execution time or power consumption. There are three levels of mutation applied in the selected mapping listed in consecutive order:

Step 4a: The first level of mutation refers to Communication objects remapping based on the memory access time. Communication objects mapped on HW memories (local or remote) with the highest access time, are remapped to memories with less or the least access time (local memories), in order to achieve less communication time provided by the mutated mappings.

Step 4b: The second level of mutation consists of (1) the identification of the component with the highest execution time that is parallelizable, (2) splitting it into two subcomponents, (3) generation of random mappings based on the modified component network. This operation aims on increasing the component parallelism in the mutated mappings in order to decrease the total computation time.

Step 4c: The third level of mutation targets the power consumption optimisation. It consists of (1) the identification of the component with the highest power consumption, (2) the move in a neighbouring hardware element and (3) the generation of random mappings for the rest of application components. This operation aims on investigating other mapping alternatives for less power consumption.

Step 5: The mutation loop is executed to populate the Pareto optimal solutions, until the user defined number of iterations is reached or until mutation process cannot provide better mappings for several executions.

Finally, the Multi-Objective Mapper produces the optimized deployment plan consisting of two decision vectors indicating the mapping of the components to the platform resources:

The first decision vector includes the mapping of application components to corresponding hardware processing elements:

$$\boldsymbol{Map^c\ X} = \ < \boldsymbol{X_1},\ \ldots,\ \boldsymbol{X_N} > \tag{35}$$

with $X_i \in X: i \in N$, and $X_i \in [1,\ C + F + G + V]$, where N the number of application components.

The second decision vector includes the mapping of communication object to physical memories as described by the following formula:

$$\boldsymbol{Map^m\ Y} = \ < \boldsymbol{Y_1},\ \ldots,\ \boldsymbol{Y_M} > \tag{36}$$

with $Y_i \in Y: i \in M$, and $Y_i \in [1, CM + FM + GM + VM]$, where M the number of communication objects.

We need to note that the initial vector will include the one given (if available) by the application developer in the System Configuration. In addition, constraints may be applied in order not to change specific mappings.

2.3.1 Early/Full Prototypes functionality

First Version for MOM's Full Prototype:

The version of the Generic MOM full prototype is covering non-functional requirements focusing on time performance (both computation and communication), power consumption, as well as memory utilization. Data movement is considered by the minimization of the communication time. Functional requirements are fulfilled by the execution of the application as defined by the user. All three use cases are currently supported. The current version of the Generic MOM supports in full extent CPU-GPU based architectures and CPU-FPGA based architectures, as defined in the problem formulation described in Section 2.2.2.1 (equation (5)). This full version integrates security, by taking as input the component network as derived by the application of the Component Network Execution Integrity (see 5.3.1). For GPUs and if high security is required, MOM avoids the use of the GPU from the component to be secured from exfiltration and assigns an FPGA instead. If the GPU is to be used, MOM ensures that the only component using it is the one that is being secured, and prevents all other components to be in the same GPU. For CPUs, PHANTOM relies on the host operating system of the involved CPUs for the implementation of execution integrity, while for FPGAs the hardware interfaces between the CPU to the FPGA are manipulated by the associated CPU process and the operating system, but the FPGA hardware and its configuration through implemented in WP4 FPGA-specific mechanisms (further described in D4.3). Further details can be found in Sections 5.2 and 5.3.1. Finally, the use of the Repository to perform interactions guarantees for security and authentication for PHANTOM applications.

Updated Version of Full Prototype and Next Steps:

The final steps of the work will be carried out as part of the integration activities in WP5. These steps include the refinements of the implemented final full version of the Generic MOM based on evaluation of MOM in PHANTOM use cases and the further elaboration and improvements based on the requirements listed in Section 2.1.1.

In addition, the next version of the Generic MOM will consider extra non-functional properties, including:

- Thermal, given that there are means to be monitored for the specific processing element from the Monitoring Framework or derived with the use of data analytics.
- Dependability, by taking into account the outcome provided by MBT functional and/or non-functional testing.

Apart from use case integration, the Generic MOM will also be integrated with the Offline MOM (see Section 3) and MBT-MOM strategy implementation (see Section 5.2.1.4).

2.3.2 Testing results

The current general MOM implementation is tested mainly in terms of achieved execution time of the user application, which is also considered as the KPI 1.3: Overall performance improvement, defined in D5.1 section 3.4.1 KPIs, which is measured by comparing the execution time of the same tasks between the current implementation of use case application and the PHANTOM optimized implementation. The application that was developed for evaluation purposes consists of two components connected via a communication object. An executed test is provided in the following figure:

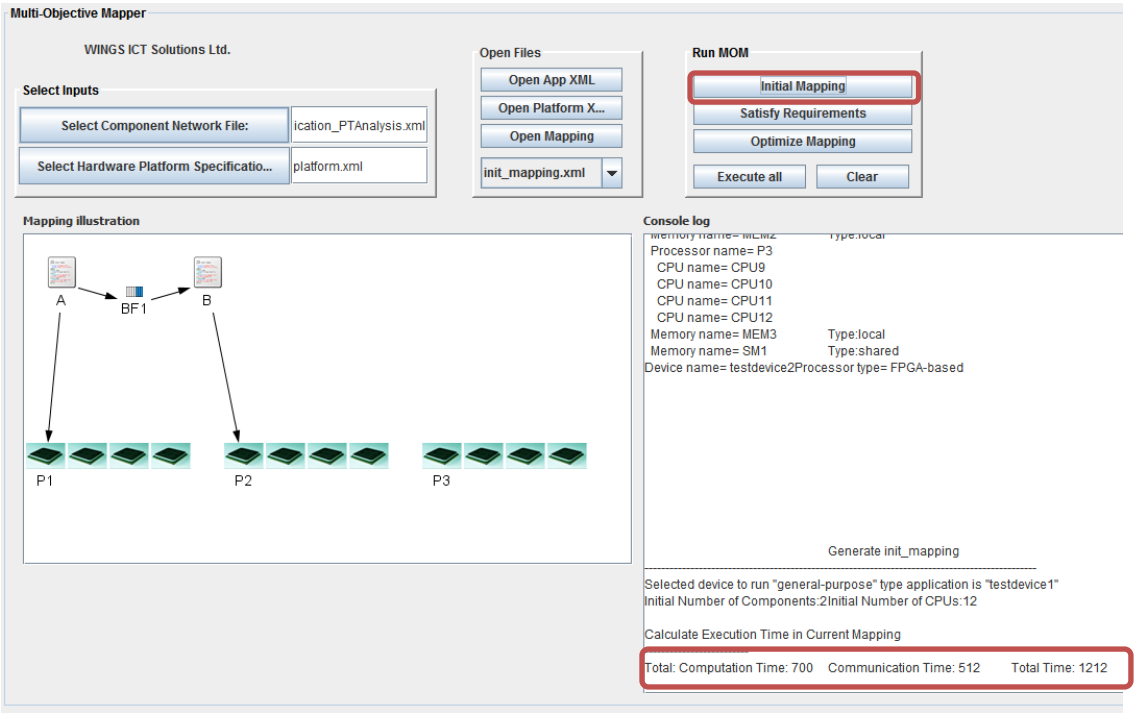


Figure 9: Generic MOM evaluation-GUI along with initial mapping

As depicted, the underlying platform consists of a multicore CPU platform. The initial mapping indicates that component "A" should be mapped to a CPU of the multicore processor "P1", while the component "B" should be mapped on a CPU of the processor "P2".

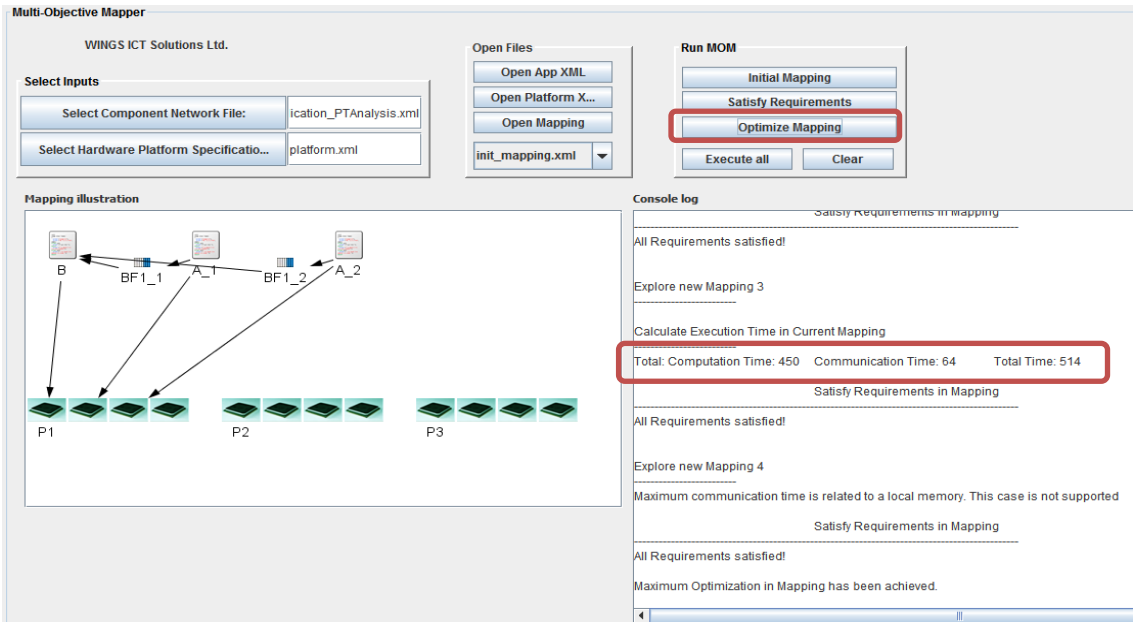


Figure 10: Generic MOM evaluation-Optimized mapping along with results

The above figure depicts the MOM outcome, which is the optimized deployment. In this example, MOM splits component "A" into two subcomponents in order to be executed in parallel and mapped to the same multicore processor in order to minimize the communication overhead. The total execution time in this case was reduced by 35%. Further results will provide the progress on MOM refinements and on the experimentation with the PHANTOM use cases.

Testing Results on the Surveillance use case

Considering all three target hardware platforms (CPU, GPU, FPGA), MOM is capable of generating the best mappings for each individual hardware platform. In Figure 11, we depict part of the deployment plan of the Surveillance application running on a CPU-SMP based device. We list the mapping of four individual application components to specific CPUs on the target hardware.

```
<deployment name="Example_Mapping" xsi:noNamespaceSchemaLocation="./map_phantom.xsd">

<!--            Estimated Execution Time:844    P3:77.0mW   P1:112.5mW   P2:89.0mW   P4:29.5mW   -->
<!--            Component Deployment            -->
<target-application name="Surveillance"/>
<target-hw-platform name="CPU-SMP device"/>
<mapping name="component_Mat_map" type="processing">
    <component name="Mat" comp_id="0" subcomponents="8"/>
    <CPU processor-name="P3" device-type="CPU-SMP" CPU-type="SMP"/>
</mapping>
<mapping name="component_Vec_map" type="processing">
    <component name="Vec" comp_id="1" subcomponents="8"/>
    <CPU processor-name="P1" device-type="CPU-SMP" CPU-type="SMP"/>
</mapping>
<mapping name="component_ShipDetection_map" type="processing">
    <component name="ShipDetection" comp_id="2" subcomponents="8"/>
    <CPU processor-name="P2" device-type="CPU-SMP" CPU-type="SMP"/>
</mapping>
<mapping name="component_UpdateShipReport_map" type="processing">
    <component name="UpdateShipReport" comp_id="3" subcomponents="8"/>
    <CPU processor-name="P2" device-type="CPU-SMP" CPU-type="SMP"/>
</mapping>
```

Figure 11 Deployment plan of the Surveillance application on a CPU-SMP device

Similarly, in Figure 12, we provide an indicative part of the deployment plan of the Surveillance application running on a CPU-GPU based device. We underline that in the deployment plan below, we only list the mapping of four individual application components to specific CPUs on the target hardware, but more importantly, we identify the loops of code in specific application components that will be executed in the graphics processing unit (GPU), which in this case is the *MaxLocal2D*.

```
<deployment name="Example_Mapping" xsi:noNamespaceSchemaLocation="./map_phantom.xsd">

<!--            Estimated Execution Time:716    P3-CPU:79.0mW   P2-CPU:99.5mW   P4-CPU:62.5mW   P1-CPU:67.0mW   -->
<!--            Component Deployment            -->
<target-application name="Surveillance"/>
<target-hw-platform name="CPU-GPU device"/>
<mapping name="component_ComputeConfidence_map" type="processing">
    <component name="ComputeConfidence" comp_id="0" subcomponents="8"/>
    <CPU processor-name="P3-CPU" device-type="CPU-GPU" CPU-type="SMP"/>
</mapping>
<mapping name="component_IDWT2_map" type="processing">
    <component name="IDWT2" comp_id="1" subcomponents="4"/>
    <CPU processor-name="P2-CPU" device-type="CPU-GPU" CPU-type="SMP"/>
    <loop-mapping id="1" target-hw-element="CPU" loop-iterations="8"/>
</mapping>
<mapping name="component_dwt2_map" type="processing">
    <component name="dwt2" comp_id="2" subcomponents="8"/>
    <CPU processor-name="P3-CPU" device-type="CPU-GPU" CPU-type="SMP"/>
    <loop-mapping id="0" target-hw-element="CPU" loop-iterations="4"/>
</mapping>
<mapping name="component_MaxLocal2D_map" type="processing">
    <component name="MaxLocal2D" comp_id="3" subcomponents="8"/>
    <CPU processor-name="P4-CPU" device-type="CPU-GPU" CPU-type="SMP"/>
    <loop-mapping id="0" target-hw-element="GPU" loop-iterations="4"/>
</mapping>
```

Figure 12 Deployment plan of the Surveillance application on a CPU-GPU device

Finally, in Figure 13, we provide an indicative part of the deployment plan of the Surveillance application running on a CPU-FPGA based device. In the following deployment plan, we only identify the loops of code in specific application components that will be executed in the dedicated FPGA IP cores, which in this case are the *IDWT2* and *DWT2*. The rest of the Surveillance application components are mapped to the rest

of the CPU-FPGA device processors as well (not only P3 as indicated below in the partial deployment plan).

```
<deployment name="Example_Mapping" xsi:noNamespaceSchemaLocation="./map_phantom.xsd">

<!--            Estimated Execution Time:700    P3-CPU:166.25mW   P4-CPU:109.0mW   P1-CPU:27.25mW   P2-CPU:5.5mW   -->
<!--            Component Deployment            -->
<target-application name="Surveillance"/>
<target-hw-platform name="CPU-FPGA device"/>
<mapping name="component_IDWT2_map" type="processing">
    <component name="IDWT2" comp_id="0" subcomponents="4"/>
    <CPU processor-name="P3-CPU" device-type="CPU-FPGA" CPU-type="SMP"/>
    <loop-mapping id="1" target-hw-element="FPGA" loop-iterations="8"/>
</mapping>
<mapping name="component_dwt2_map" type="processing">
    <component name="dwt2" comp_id="1" subcomponents="8"/>
    <CPU processor-name="P3-CPU" device-type="CPU-FPGA" CPU-type="SMP"/>
    <loop-mapping id="0" target-hw-element="FPGA" loop-iterations="4"/>
</mapping>
<mapping name="component_MaxLocal2D_map" type="processing">
    <component name="MaxLocal2D" comp_id="2" subcomponents="8"/>
    <CPU processor-name="P3-CPU" device-type="CPU-FPGA" CPU-type="SMP"/>
    <loop-mapping id="0" target-hw-element="CPU" loop-iterations="4"/>
</mapping>
<mapping name="component_Simons_map" type="processing">
    <component name="Simons" comp_id="3" subcomponents="8"/>
    <CPU processor-name="P3-CPU" device-type="CPU-FPGA" CPU-type="SMP"/>
</mapping>
```

Figure 13 Deployment plan of the Surveillance application on a CPU-FPGA device

To compare all three complete deployment plans (Figure 11-13 only show part of the deployment plans), we have summarized the results of total execution time and power consumption in the table below (Table 1). Based on the obtained results, we conclude that the application mapping on an FPGA provides 17% less execution time than the one on a CPU-based device. Power consumption distribution on the different processors is also depicted for all target hardware platforms. These are indicative results as the complete report on the Generic MOM results in all three use cases will be delivered as part of the integration activities in WP5.

Table 2: Performance Analysis results of Surveillance application deployment plans

Surveillance Application	Total Execution Time (ns)	Power Consumption (mW/processor)			
		P1	P2	P3	P4
CPU-SMP	844	77.0	112.5	89.0	29.5
CPU-GPU	716	79.0	99.5	62.5	67.0
CPU-FPGA	700	166.25	109.0	27.25	5.5

2.4 INTEGRATION ASPECTS

In order to provide the best mapping, MOM interacts with a multitude of PHANTOM tools positioned in the PHANTOM design flow (see Figure 2 and in Section 2.2). A more detailed representation of MOM's positioning in the PHANTOM tool-flow is found in Figure 2 in Section 2.2 as well. First, MOM requires the component network, the number of application components and the application components' source code along with the communication objects. These are provided by the user-defined documents that are stored inside the Repository through the Programming model.

Similarly, MOM requires the System configuration, an optional Initial Mapping, the System Model along with user/use case requirements from the Repository through the Programming model. Furthermore, a specific MBT-MOM interaction has been designed and implemented to collect performance metrics by MBT and produce a pool of initial mappings for MOM. MOM interacts with the Model Based Testing in order to retrieve sets of estimated/measured performance metrics of the application or per individual component, which facilitates the MOM's optimization process towards a more focused pool of solutions. In addition, MOM interacts with the monitoring framework to acquire the latest monitored and analysed performance metrics, for further optimization towards the actual monitored data. An example of application performance metrics targeting computation time and power consumption is depicted in Figure 14 below.

```
<MF-monitored-computation-time measurement-unit="ns" value="34"/>
<MF-monitored-power-consumption measurement-unit="milliwatt" value="12"/>
```

Figure 14: Example of application performance metrics

Finally, MOM will require the parallelization directives from the Parallelization toolset that further assist MOM to decide on the number of component's parallelization. In Figure 15 below, an example of parallelization directives is found. The example specifies for a given loop in an application component, whether it is parallelizable or not, the maximum number of loop iterations, and whether it is FPGA and/or GPU compatible or not. This information will be considered by MOM to derive the deployment plan of the specific application component.

```
<!--Parallelisation Directives-->
<loop FPGA-compatible="false" GPU-compatible="true" id="0" loop-iterations="4" parallelizable="true"/>
```

Figure 15: Example of Parallelization Directives

In the following Figure 16 and Figure 17, we specify the XML description of the platform specification for CPU-GPU and CPU-FPGA devices that are used in the context of the execution stage of the PHANTOM tool-flow. In Figure 16, we assume a CPU-GPU device with a processor containing a four CPUs layout and a GPU with the capacity of four distinct multiprocessors. Respectively, in Figure 17, we assume a CPU-FPGA device with a processor containing a four ARM-based CPUs, and a FPGA with the capacity of various predefined IP cores.

```xml
<!-- CPU-GPU based Device Description -->
<device name="CPU-GPU device" type="CPU-GPU">

    <processing-node name="unit1" type="CPU-SMP" architecture="SMP">
        <processor name="P1-CPU" type="INTELX">
            <configuration name="core number" value="4"/>
            <configuration name="cpu frequency" value="X" unit="GHz"/>
            <configuration name="bytespercycle" value="1"/>
            <memory name="MEM1" size="512" size-unit="MB" access-time="1" access-time-unit="ns/word"/>
        </processor>

        <processor name="P2-CPU" type="INTELX">

        <processor name="P3-CPU" type="INTELX">

        <processor name="P4-CPU" type="INTELX">

    </processing-node>

    <processing-node name="unit2" type="GPU" brand="NVIDIA">
    <!-- GPU specification Defined in Surveillance Use case -->
        <multiprocessor name="mpc1" type="multiprocessor" blocks="2" threadsperblock="1024">
            <memory name="mpc1_blockmem1" type="blockmemory" value="48" units="KB"/>
            <memory name="mpc1_blockmem2" type="blockmemory" value="48" units="KB"/>
        </multiprocessor>
        <multiprocessor name="mpc2" type="multiprocessor" blocks="2" threadsperblock="1024">
        <multiprocessor name="mpc3" type="multiprocessor" blocks="2" threadsperblock="1024">
        <multiprocessor name="mpc4" type="multiprocessor" blocks="2" threadsperblock="1024">
    </processing-node>

    <local_bus name="PCI1" type="PCIe" throughput="15" throughput-time-unit="Bytes/ns"/> <!--15GB/s-->

    <comm_interface name="EXT1" type="Ethernet Network">
        <configuration name="speed" value="100" units="GBit/s" ip="192.168.1.1"/>
    </comm_interface>

    <memory name="SM1" type="RAM" size="16384" size-unit="MB" access-time="8" access-time-unit="ns/word"/>

</device>
```

Figure 16: XML Description of a CPU-GPU based device

```xml
<!-- FPGA based Device Description -->
<device name="CPU-FPGA device" type="CPU-FPGA">

    <processing-node name="unit1" type="CPU-SMP" architecture="SMP">
        <processor name="P1-CPU" type="ARM-Cortex">
            <configuration name="core number" value="4"/>
            <configuration name="cpu frequency" value="X" unit="GHz"/>
            <configuration name="bytespercycle" value="1"/>
            <memory name="MEM1" size="512" size-unit="MB" access-time="1" access-time-unit="ns/word"/>
        </processor>

        <processor name="P2-CPU" type="ARM-Cortex">

        <processor name="P3-CPU" type="ARM-Cortex">

        <processor name="P4-CPU" type="ARM-Cortex">
    </processing-node>

    <processing-node name="unit2" type="FPGA" brand="Xilinx">
        <fpgalogic name="PL1" type="xc7z020clg400">
            <resource name="LUT" type="lookuptables" maxvalue="53200"/>
            <resource name="LUTRAM" type="lookuptablesRAM" maxvalue="17400"/>
            <resource name="FF" type="flipflop" maxvalue="106400"/>
            <resource name="BRAM" type="blockRAM" maxvalue="140"/>
            <resource name="DSP" type="digitalsignalprocessing" maxvalue="220"/>
            <resource name="BUFG" type="bufferglobal" maxvalue="32"/>
            <configuration name="maxfrequency" value="800" units="MHz"/>
            <memory name="SM4" type="DDR3" size="1024" size-unit="MB" access-time="8" access-time-unit="ns/word"/>
        </fpgalogic>
    </processing-node>

    <local_bus name="AXI" type="AXI4" throughput="15" throughput-time-unit="Bytes/ns"/> <!--15GB/s-->

    <comm_interface name="EXT3" type="Ethernet Network">
        <configuration name="speed" value="100" units="GBit/s" ip="192.168.1.3"/>
    </comm_interface>

    <memory name="SM1" type="RAM" size="16384" size-unit="MB" access-time="8" access-time-unit="ns/word"/>

</device>
```

Figure 17: XML Description of a CPU-FPGA based device

2.4.1 Model Based Testing - Multi-Objective Mapper Interaction

MBT - MOM strategy collects the performance metrics of application components on representative hardware and sends the information to MOM as a mapping reference. The objective of the specific MBT-MOM interaction is to automatically execute applications and individual components mandatorily over representative environments, collect performance metrics and send the collected information to MOM as initial mapping reference. In MBT – MOM strategy, MBT indicates the hardware over which a component is expected to be executed by providing with deployment manager a deployment plan and overwrites the deployment plan generated by MOM; after the execution of the components over target hardware, the related performance metrics are collected regarding each component and sent back to MOM.

The MBT implementation details are introduced in “D3.2 - Final report on programmer- and productivity-oriented software tools”. Generally, the MBT in PHANTOM consists of four activities in two phases i.e., early validation and test execution. The MBT-MOM interaction follows the same process of the creation of MBT models, test generation and test concretization in MBT workflow. Based on the activity of non-functional testing in test execution phase, the MBT - MOM strategy additionally generates deployment plans to force the execution of individual components over different hardware; sends specific deployment plans to deployment manager; triggers the execution by obliging that the

execution follows the specified deployment plan. At the end of execution, the performance metrics are collected for different deployment and sent back to MOM for execution.

The MBT-MOM strategy consists of a list of steps to collect performance metrics for MOM, which are described as follows:

Step 1. MBT firstly gets from Repository a) platform description, and b) component network.

Step 2. MBT generates a list of deployment plans based on the received files from last step. Particularly, each deployment plan contains one component to deploy and the target hardware to be deployed. The deployment plan generated by MBT is different from the one provided by MOM, and overwrites the MOM generated deployment plan. The reason is that instead of executing a component in an optimized manner, the deployment plan from MBT is to force the execution of certain component over representative hardware so as to collect performance metrics, even if the target hardware is not optimal from performance perspective.

Step 3. MBT sends input data and deployment plan to the Repository. The deployment plan overwrites the deployment plan generated by MOM if any, so that the components are executed over different hardware instead of only optimal ones.

Step 4. MBT sends the start trigger of execution to the Application Manager, and the platform starts the execution of application/components.

Step 5. Once the execution is over, the Application Manager sends the end signal to MBT via notification.

Step 6. MBT gets the output data from the Repository. In case the output data is too large (e.g. a large image), the location where output data is stored will be sent back instead. The execution fails if the executed component/application does not produce the expected results; on the contrary, the performance will be collected as shown in the following steps if the functional outputs are the same as expected.

Step 7. MBT gets performance information from the Execution Manager.

Step 8. MBT sends non-functional testing results to the Repository so that MOM can get the performance metrics.

Figure 18 illustrates one individual deployment plan to execute specific component A_1 over the hardware CPU2; Figure 19 presents the collected execution time for component A_2 over two hardware CPU3 and GPU1.

```
<deployment name="Initial_Deployment" xsi:noNamespaceSchemaLocation="./map_phantom.xsd">
    <mapping name="component_A_1_map" xsi:type="processing" persistency="restricted">
        <mapping_restriction target_type="CPU"/>
        <mapping_restriction target_type="GPU"/>
        <component name="A_1" />
        <processor name="P1" cpu-name="CPU2"/>
    </mapping>
</deployment>
```

Figure 18 Individual Deployment plan for specific component

```
<metrics name="Performance_Metrics_A_2" xsi:noNamespaceSchemaLocation="./metrics_phantom.xsd">
    <mapping name="component_A_2" xsi:type="processing" persistency="full">
        <component name="A_2" />
        <processor name="P1" cpu-name="CPU2"/>
        <execution_time value="1781">
    </mapping>
    <mapping name="component_A_2" xsi:type="processing" persistency="full">
        <component name="A_2" />
        <processor name="P4" gpu-name="GPU1"/>
        <execution_time value="972">
    </mapping>
</metrics>
```

Figure 19 Performance Metrics for execution time

2.5 SUMMARY OF INNOVATIONS BEYOND SOTA

The Generic MOM is based on a novel approach inspired by multi-objective and bio-inspired paradigms. Generic MOM innovation relies on the following features:

- Multi-dimensional optimization: against multiple objectives and targeting systems with increased heterogeneity (CPU, CPU-GPU, CPU-FPGA) across the computing continuum (from embedded to compute clusters).
- Optimization process assisted by Model Based Testing for estimation and verification of application's requirements satisfaction and Monitoring Framework to monitor both resources and application.

2.5.1 Background technologies utilised in development

The generic MOM does not rely on existing tools and developments, but is a development from scratch.

2.5.2 Advances of our implementation

A novel problem formulation (described in Section 2.2.2.1) has been developed from scratch to address the idea of multi-dimensional optimization in terms of optimizing against multiple objectives and targeting systems with increased heterogeneity (CPU, CPU-GPU, CPU-FPGA) across the computing continuum (from embedded to compute clusters) in full consistency with the PHANTOM Component-based Programming Model.

Based on this problem formulation, MOM has been developed using Java Programming language and Eclipse platform as Integrated Development Environment, using appropriate XML libraries to be able to parse its input and produce its output. MOM is inspired by multi-objective optimization (as depicted in equation 25, several objectives can be considered) and bio-inspired paradigms (as described in Section 2.3 a genetic algorithm is developed; the genetic algorithm is a metaheuristic inspired by the process of natural selection and which relies on bio-inspired operations such as mutation and selection). Generic MOM implementation consists of 5 steps executed in iteration (described in Section 2.3), which have been implemented thoroughly in the scope of PHANTOM and optimized for addressing PHANTOM requirements.

3. THE PHANTOM OFFLINE MOM

3.1 REQUIREMENTS

The following subsections discuss the status of the requirements, which consist of two sets:

- The initial set of requirements imposed by the use case providers at the beginning of the project and specified in Deliverable D1.1
- Additional requirements that were obtained from the feedback of use case providers, done after the release of preliminary version of PHANTOM tools by M18.

3.1.1 Initial Requirements

The Offline MOM is motivated by the requirements of the Telecoms and Surveillance use cases to be able to create and verify applications with real-time requirements. Whilst the main PHANTOM toolchain, supported by the Generic MOM and the monitoring framework, can focus on timing and verify that requirements are met, this requires deployment and measurement-based testing. The Offline MOM attempts to use real-time analysis to improve the reliability of applications, decrease development time, and remove the requirement for application developers to have any real-time expertise.

Considering the requirements described previously in Deliverable D1.1, this focuses on the optimisation requirements U49, U50, U54, U55, U57, U60, U61 and U63.

Table 3: Initial requirement to Offline MOM

ID	Requirement	Priority	Status
U49	The mapping proposed by the PHANTOM framework shall reason about the functional and non-functional requirements of the application	SHALL	Achieved
U50	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for execution time	SHOULD	Achieved
U54	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for communications bandwidth	SHOULD	Achieved
U55	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for I/O	SHOULD	Achieved
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	Achieved

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	Achieved
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	Achieved
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	Achieved

Note that I/O (U55) is currently only considered in terms of availability (a given component must be deployed into a given location) or bandwidth (for buses and networks that support bandwidth analysis).

3.1.2 Preliminary Feedback from M18 Results

Table 4: Additional requirements to Offline MOM

ID	Requirement	Priority	Status
AU1	Offline MOM shall interact with the Application Manager to start its operation	SHALL	Achieved
AU2	Offline MOM shall interact with the Repository for its input	SHALL	Achieved
AU3	Offline MOM shall write its results to the Repository so that the Generic MOM and Deployment manager may use it transparently	SHALL	Achieved

The main feedback from M18 was that the Offline MOM must be able to operate transparently within the PHANTOM framework. As its main goal is to reject infeasible mappings, it should be able to do this without direct user interaction. Indeed, unless specifically executed it should appear to be a part of the Generic MOM.

This has been achieved thanks to the work of the Repository and the Application Manager. The Offline MOM subscribes to the development activities of the current project, and automatically executed when Deployments are added to the Repository.

3.2 DESIGN SPECIFICATIONS

Recall that the PHANTOM MOM has three inputs:

1. System Model (Component Network) – describes the components in the application, the shared data elements, and how they are all connected.

2. Platform Model (Platform Description) – describes the target platform architecture in broad terms, such as the CPUs and communication channels that exist.

3. System Configuration (Deployment) – Describes how the components of the application are mapped to the processing elements of the platform, and how the shared data elements are mapped to the memories of the system.

The developer of the system will constrain some elements of the deployment initially, but many deployment options may still be available. The role of the Generic MOM is to assign all unconstrained mappings to create a complete deployment, which it does through the use of an evolutionary/bio-inspired multi-objective, genetic driven approach that relies on profiling data and objectives (execution time, power, etc.) estimates to pick a mapping which is likely to perform well. It then tests this at runtime by making use of the PHANTOM monitoring framework. If the deployment does not perform adequately well, then the process can be repeated. The Offline MOM aims to accelerate this process by using analytical approaches to determine ahead-of-time the timing properties of a given deployment and therefore eliminate mappings that fail to meet design constraints.

Timing analysis is not yet mature enough to be able to perform worst-case response time analysis over the most complex multicore systems with on-chip networks. The PHANTOM approach is therefore aimed at a development flow that can integrate the current state-of-the-art, and evolve to use newer techniques as they become available. It is for this reason that the primary function of the Offline MOM is as a necessary condition, rather than a sufficient one. i.e. It can *reject* mappings, but only *verify* mappings when the target platform is simple enough such that there is a suitable analytical approach. The PHANTOM approach is designed to be able to adapt as state-of-the-art analysis techniques advance in order to handle more complex architectures.

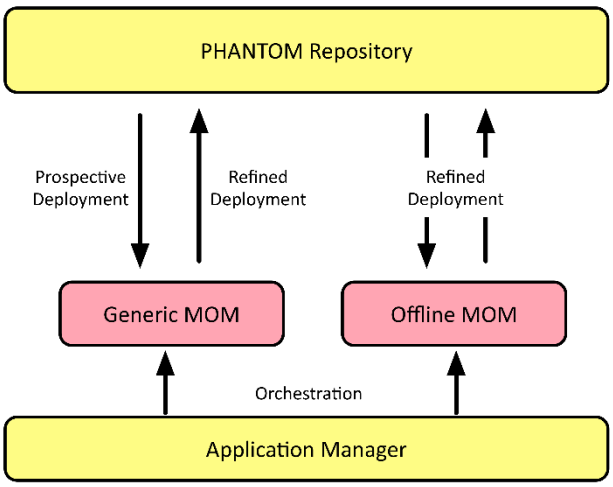


Figure 20: The Offline and General MOM toolflow

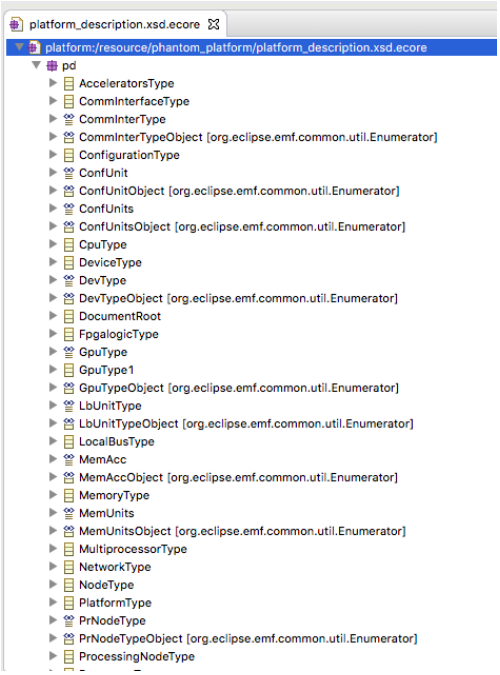
Figure 20 shows how the Offline MOM integrates into the PHANTOM platform. The initial deployment is created by the developer and contains the elements of their design that they already know to be fixed. The Generic MOM is then used to refine this into a fully-specified deployment, in which all components are speculatively mapped. The Application Manager will then invoke the Offline MOM to analyse this deployment to verify whether problems are likely to exist with the mapping. If not, then the developer proceeds to deployment. If problems are found, a set of constraints are added to the deployment to exclude the current deployment, and the Generic MOM is invoked again to attempt another mapping.

This flow does not restrict the system designer from creating the system that they wish. When a timing analysis approach exists for the system that they are working with, it will automate the application of that approach. For example, if the target platform is a virtualized HPC cluster then currently no exact timing analysis exists and so the Offline MOM will not be able to help. However, if a symmetric multiprocessing system is specified with a suitable real-time OS, then a wealth of existing work can be applied [5]. It is the work of the Offline MOM to identify the scenarios when existing analysis can be applied.

3.3 IMPLEMENTATION DETAILS

3.3.1 Metamodelling

The Offline MOM is powered by a metamodelling approach based on the Eclipse Modelling Framework [6] and Epsilon [7]. The Component Network, Platform Description, and Deployment are all described by a consistent Ecore metamodel in the Eclipse Modelling Framework. This allows PHANTOM to use the Epsilon project to easily create model comparison and transformation toolchains. Ecore is a widely-used, open metamodelling standard so it is possible to use this broader in the project. A fragment of this model is shown in Figure 21.




```
<platform xmlns="PD"
    name="Example_Platform"
    noNamespaceSchemaLocation=
      "../models/platform_description.xsd">
  <device name="testdevice1" type="CPU">
    <processing-node name="unit1" type="CPU-based">
        <processor name="P1" type="INTELX">
           <cpu name="CPU1"/>
           <cpu name="CPU2"/>
        </processor>
    </processing-node>
  </device>
</platform>
```

Figure 21: **A fragment of the Ecore metamodel and an instantiation in XML format**

All inputs to the MOM must be valid instantiations of this metamodel.

3.3.2 Model Pattern Matching

As described earlier, real-time analysis cannot yet handle all of the platforms that can be described by the PHANTOM metamodel. Therefore, the approach taken by the Offline MOM is to use model pattern matching to extract parts of the model that can be analysed and focus on those areas.

The Epsilon project contains a language specifically for this purpose called Epsilon Pattern Language (EPL). EPL and Epsilon can search through the MOM's input model for parts that correspond to a set of requirements. For example, a simple pattern is as follows:

```
pattern SingleCorePreEmpt
      cpu : ProcessingNode,
      os : FieldDeclaration from cpu.ostype,
      sched : FieldDeclaration from cpu.schedtype,
      shared : Set(SharedData)  from cpu.sharedObjs,
      guard : (os.value = "rt_preempt"
            and sched.value = "fp"
            and shared.count() = 0) { }

operation sharedObjs(name : String) : Set(SharedData) {
      //Fetch all shared objects accessed by
      //software deployed to this CPU
}
```

When applied to a model, this will match all CPUs that have an operating system capable of real-time preemptive scheduling, that uses fixed priority scheduling, and that doesn't share any data with other CPUs. This can easily be analysed. A more complete version needs to also check that the assigned components have priorities assigned and that there are actually timing requirements to be checked.

Similar patterns can be generated to describe other types of analysis by, for example, modelling shared data across networks using sporadic or execution time servers. The main advantage of this approach is that it can easily extend as the state-of-the-art develops by creating new EPL patterns to match.

3.3.3 Translation to MAST

Once an area of the model has been identified by an EPL pattern that can be analysed, it is translated into an input model for MAST [8]. MAST (Modeling and Analysis Suite for Real-Time Applications) is an open-source suite of tools for performing various kinds of timing and schedulabilityanalyses . MAST contains a wide range of pre-made tools for state-of-the-art real-time analysis, so it is not necessary to re-implement all of these. The model transformation is easily implemented using the Epsilon framework's Epsilon Generation Language (EGL), a dedicated model-to-text transformation. EGL is a template-based language that can query the model to generate textual output. An example transformation for describing the CPUs of the platform for MAST is as follows:

```
[%
for(cpu in "phantom_processor".getAllInstOfStereotype()) {
%]
           Processing_Resource (
                 Type => [%=cpu.type%],
                 Name => [%=cpu.name%],
[%
      if(cpu.worst_context_switch.isDefined()) {
%]
                 Worst_Context_Switch =>
                       [%=cpu.worst_context_switch%]
           );
[%
      }
}
%]
```

The code traverses an Ecore model and outputs a MAST input model. In the above example, every instance of the `phantom_processor` stereotype in the platform description creates a `Processing_Resource` in the MAST model with its parameters set from the model.

Each EPL pattern must invoke an associated EGL translation so that the framework will be able to automatically match areas of the model of interest, translate to MAST, and invoke MAST to perform the analysis.

Once the MAST translation is complete, the Offline MOM invokes MAST on the translated model to check schedulability. If MAST determines there is a problem with the deployment (if a timing requirement might be violated) then another deployment must be generated - it is not necessary to deploy and test. The initial deployment is amended to make the current deployment invalid and the Generic MOM is re-invoked.

3.4 INTEGRATION ASPECTS

In order to facilitate the transparent use of the Offline MOM, it is integrated with the PHANTOM repository through the use of the Application Manager. The Offline MOM subscribes to a project using the Application Manager's websocket interface. This will launch the Offline MOM every time the deployment metadata for the project is updated.

The Application Manager allows different kinds of notifications. Currently the Offline MOM is triggered every time a new file is added into the current project, however as some analyses can require significant computation, it is important to avoid repeated work. To achieve this, the Repository stores metadata with each file that is used by the tools of the platform to store ancillary information. The Offline MOM uses this metadata to note which deployments it has already checked. A file may have the following metadata:

```
{
      "project": "project_name",
      "source": "user",
      "data_type": "deployment",
      "checked": "no"
}
```

The Offline MOM fetches each unchecked deployment from the Repository. Deployments contain references to the Component Network and Platform Description for which they are created. These files are then fetched and the validation is executed. If a deployment has failed, then the metadata is updated to include an error string that can be relayed to the user.

Once its work is complete, the Offline MOM updates the metadata of the files it has checked. This can then be used to trigger further downstream tools because the Application Manager will notify any tools that have subscribed accordingly.

3.5 SUMMARY OF INNOVATIONS BEYOND SOTA

This work does not intend to create new analysis techniques. The primary innovation in this approach is the seamless integration of the state-of-the-art into the development process in a way that allows the value of that research to be used by non-experts. The platform automatically makes use of the Offline MOM to reject failing mappings early, thereby saving development time.

The Component-based Programming Model was defined in order to assist with real-time analysis. By requiring applications to be componentised and their communications explicitly elaborated, it is possible to analyse the system in parts. This is an advance over traditional real-time analysis, which attempts to consider the entire system as a

whole. This still relies on a communications platform/protocol that has determinable worst-case performance guarantees.

4. PHANTOM MONITORING SERVER AND EXECUTION MANAGER

The architecture of the PHANTOM run-time Monitoring Framework (MF) plays an essential role in the application optimization based on the understanding of both software non-functional properties and hardware quality attributes with regard to performance and energy consumption aspects. The architecture of MF is composed of the Monitoring Library, the MF-Server, and the MF-Client, interconnected as shown in Figure 22.

The MF-server main task is to receive and to store data (such as metrics and details of which application and where it ran). The data is collected at the system level by the MF-Client, and at the application level by the applications with the MF-Library. The Monitoring Server and the Execution Manager both provide the functionalities to query and analyse the registered data. However, the Execution Manager's main task is to focus on analysing the data from executed applications.

The object of the implementation of the Execution Manager is to avoid latencies due to the intensive input communication traffic in the Monitoring Server when registering data.

The Execution Manager and the Monitoring Server are both intended to work on the place where the data is stored because the efficiency of the system improves by performing the data analytics at the place where the data is stored, rather than sending a large amount of data to be processed in another location.

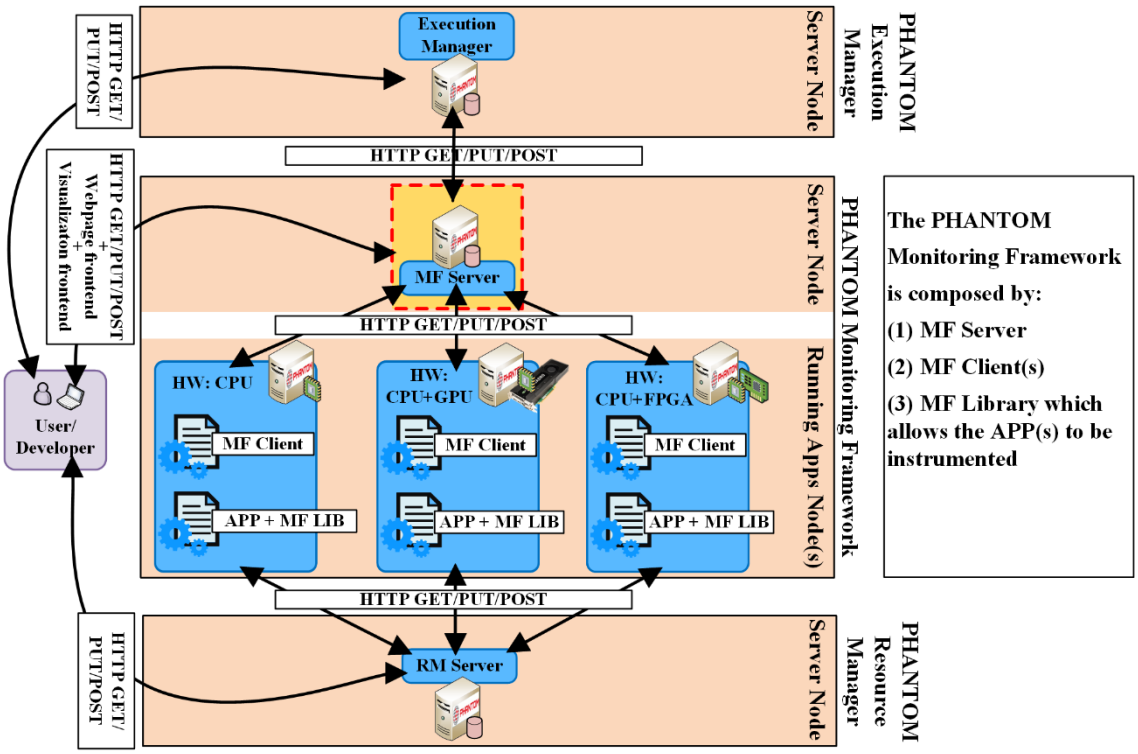


Figure 22. PHANTOM Monitoring Framework Architecture. In the figure is highlighted MF server.

The following section discusses design specifications and the status of the requirements. After that, the interfaces, dependences, and integration with the PHANTOM toolset are presented. Finally, the major innovations implemented in the MF-server are identified.

The MF-Library was integrated with the MF-Server under task 2.2 "Unified runtime monitoring implementation". However, we consider that MF-Library API, targeting to enable the collection of metrics at application level in a user-defined manner, has to be described in D3.2 because is there where are described the PHANTOM APIs ("Programming interfaces (APIs) per application class" task 3.2). This allows having all the tools used for the instrumentation of the applications in one place, as well as helps to provide a more clear view on the purpose and future use of the MF-Library API.

4.1 REQUIREMENTS

The requirements, which consist of two sets:

- The initial set of requirements, imposed by the use case providers at the beginning of the project and specified in Deliverable D1.1. The requirements in the Monitoring Framework can be related to one or more components of the MF architecture.

- Additional requirements that were obtained from the feedback of use case providers, done after the release of a preliminary version of PHANTOM tools by M18.

In the following, we describe the initial requirements (indexed as Uxx), and the new requirements detected from the questionnaires filled by the use case partners (indexed as Nx).

4.1.1 Initial Requirements

Here are the requirements in two different categories, first those related to accessibility, and second, those related to integration.

Accessibility (U35, U79, U81)

The run-time monitoring accessibility requirements are mainly the following:

- Data obtained by the run-time monitoring framework shall be accessible to the users by some means based on the users' interest.

- Data obtained by the users should be structured in a standard format, which enables further integration requirements.

- Users should be able to control the metrics sampling frequency and to select which metrics are to be monitored.

- Data storage and historical metrics analysis shall also be provided in the monitoring framework.

Table 5. Accessibility use case requirements addressed by MF-Server and Execution Manager

Req. No.	Requirement	Overall Priority	Status
U35	The PHANTOM framework shall be capable of interfacing with local target platforms (deploying the application, monitoring the execution and the state of the target platform resources).	SHALL	**Achieved**
U79	The data obtained by the run-time monitor shall be accessible and exposed to the user for their own tasks	SHALL	**Achieved**
U81	It should be possible to export the run-time monitoring data in a structured data format	SHOULD	**Achieved**

Integration (U57, U84, U85)

The monitoring framework shall become an indispensable feature of the PHANTOM platform with the seamless integration of the run-time monitoring data with the other components (e.g. MOM). This is achieved by means of standardised service-oriented interfaces for querying both the raw data and the analytics results for them.

Table 6. Integration use case requirements addressed by MF-Server and Execution Manager

Req. No.	Requirement	Overall Priority	Status
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 to the obtained execution data	SHALL	**Achieved**
U84	PHANTOM shall provide a facility for storing and retrieving historical profiles of the stored events and metric values	SHALL	**Achieved**
U85	PHANTOM shall provide the possibility to perform some basic analytics for the stored performance data	SHALL	**Achieved**

4.1.2 Preliminary Feedback from M18 Results

The requirement found from the questionnaires filled by the Use Case Partners (D1.3), appears in the next table.

Table 7. Additional requirements addressed by MF-Server and Execution Manager

Req. No.	Requirement	Overall Priority	Status
N4	Application-specific metrics visualization with Grafana/Kabana might be a great improvement for the HPC simulation application.	SHOULD	**Achieved**

4.2 DESIGN SPECIFICATIONS

4.2.1 Architecture

As clarified in D1.2, the PHANTOM monitoring framework architecture follows the design of ATOM – the monitoring solution elaborated by the EU EXCESS and DreamCloud projects. However, the initial design of ATOM was too much infrastructure-oriented and the application-level monitoring was not supported. Therefore, ATOM has been considerably redesigned, aiming along with enabling the application-level monitoring and the more modular component-based and service-oriented architecture (see Figure 22).

The PHANTOM monitoring framework follows the client-server architecture, according to which the runtime monitoring information is collected by means of a monitoring agent service (deployed on each of the monitored hardware resources) and transmitted to a centralised service (the monitoring server) that is usually deployed on a dedicated resource.

The MF server receives the data, stores the metrics in a database, and offers the functionalities to query and analyse the run-time collected metrics. The customers of the monitoring results might be the end-users or the other PHANTOM services like MOM.

The MF server is composed of a data storage layer, used to persistently store the monitoring information from the sensors/agents, and a web-service for the data transmission from the agents to the data storage layer (cf. Figure 23). The ATOM's PHANTOM MF server is adapted according to new data query requests. Details on the queries to the monitoring server are provided in D4.2.

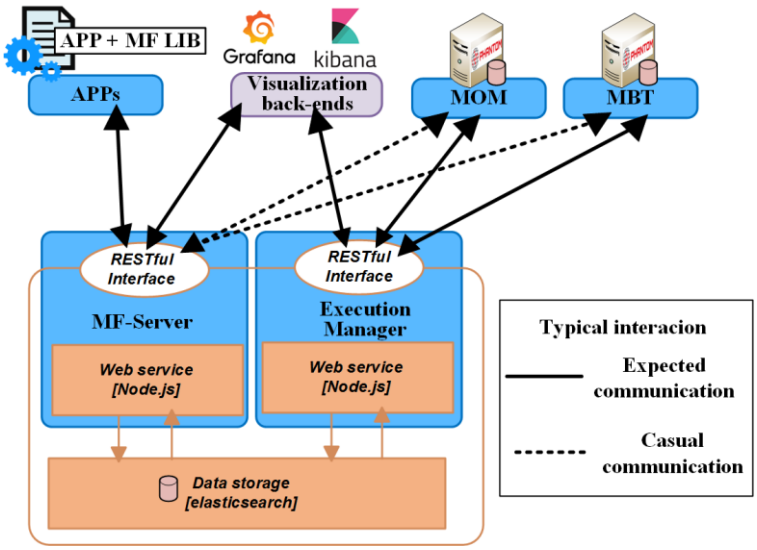


Figure 23: Workflow and components of MF server

The Web service is written in JavaScript under the Node.js runtime environment. Node.js has an event-driven architecture and is suitable to be used for data-intensive real-time applications that run across distributed devices. A free and open-source framework for Node.js – Express.js, has been selected for building the RESTful APIs.

For the data storage component, ElasticSearch[1] is used. It is a flexible and powerful open-source, real-time search and analytics engine. As a distributed, multi-tenant full-text search engine, it is preferred as supporting RESTful web interface and using schema-free JSON documents.

The communication layer

[1] For further information about ElasticSearch, please see references [11] [12] [13] [14] [15].

As with ATOM, we continue to use JSON as the primary data format for data exchange between the client and the server. It is the current trend for Web Services to be based on JSON, which is much more lightweight (compact and shorter) than XML. In addition, JSON extends the flexible data structure of XML with the possibility to define arrays.

As an example, Listing 1 shows a JSON structure sent by a client to the server at a specific time point, which is composed of a simple key-value pair (metric name and its numeric value). The time stamp for each newly arrived event is automatically generated. Following the characterization of metrics data used by time-series database, a metric is thus represented by its name and a series of numerical values collected over time. In the case of PHANTOM distributed infrastructure, the timestamps sent are revised to the local timestamps (in milliseconds), since users are more interested in time duration rather than the exact real time[2].

Listing 1: Example a typical JSON metrics

```
{
    “local_timestamp”: 1490358666276.5
    “hostname”: “CPU:node01”
    “type”: “performance”
    “core01:MFLIPS”: 415279.555
}
```

4.3 PERFORMANCE AND EVENT PREDICTIONS USING METRIC ANALYTICS

Multiple disparate data becomes available during the continuous deployment of different mappings for different applications, which can massively feed complex analytics tools for the extraction of useful deductions and meaningful knowledge. To this direction, specific algorithms are developed in order to optimally use all this information to enhance the functionalities of the PHANTOM framework.

Specifically, two problems are confronted: performance predictions in various metrics, like execution time and power based on the selected mapping’s features, and failure events detection using previous metrics as a time series to correlate their occurrence with specific events that occurred previously. Both solutions are described below, while the full versions will be analysed in D5.2 when real data will be available from the Monitoring Framework concerning the (integrated with all the PHANTOM tools) use cases.

[2] **CLOCK_REALTIME**: represents the machine's best-guess as to the current wall-clock, time-of-day time. It can jump forwards and backwards as the system time-of-day clock is changed, including by NTP.
CLOCK_MONOTONIC (referred as local-timestamp): represents the absolute elapsed wall-clock time since some arbitrary, fixed point in the past. It isn't affected by changes in the system time-of-day clock. It is the best option to compute the elapsed time between two events observed on the one machine without an intervening reboot.

4.3.1 Performance prediction using mapping features input

- ***Data manipulation***

  The Monitoring Framework is responsible to supply the user, as well as the different PHANTOM components with all kinds of metrics (that are defined in D1.2 and D1.3) from previous executions on the hardware platform. This data contributes to the MOM's selection of an optimal mapping with the development of predictive analytics on them using advanced pattern recognition techniques to predict the mappings' possible efficiency concerning performance, power and memory footprint.

  All kinds of data are used for the classification of the different mappings as well as the prediction of the metrics after the execution of specific deployment plans. In specific, previous execution times, power, I/O, and RAM metrics are correlated with the mapping features in order to create a model that is able to classify these mappings and place them in the multi-dimensional space defined by the input data.

  The data that is used as input is chosen so that it can distinguish the mappings that offer an improvement (performance, lower power, etc.) to the whole application over those that prove to be inefficient.

  Specifically, the mappings are defined by data like the following:

  - **Computation load for each hardware unit (CPU, GPU, FPGA)**: The sum of the execution times of the serial versions of all the components that are executed on each unit. This gives an indication of each unit's computation load.
  - **Number of different components run on each hardware unit (CPU, GPU, FPGA)**
  - **Communication load for each hardware unit (CPU, GPU, FPGA):** The sum of the communication objects' size that use the unit's memory. This gives an indication of the communication latency expected on each unit.

- ***Data Analysis and Classification***

  Depending on the size, the quality and the distribution of the metrics that are available from the Monitoring Framework after running different mappings for the use case applications, multiple algorithms are investigated in order to extract as much information out of the provided data as possible. In specific, the metrics' results are used for the definition of a linear or non-linear mathematical model that is able to classify a given mapping with specific features.

  Thus, after the mapping of the training data in the multi-dimensional space, a function is generated using supervised techniques that are able to distinguish the points that satisfy the use case requirements from those that do not. The algorithms that are being considered before the availability of real-world metrics include techniques like Support Vector Machines (SVMs) as well as Artificial Neural Networks (ANNs) for the creation of even more specific models (with better results) if enough data is available for their training.

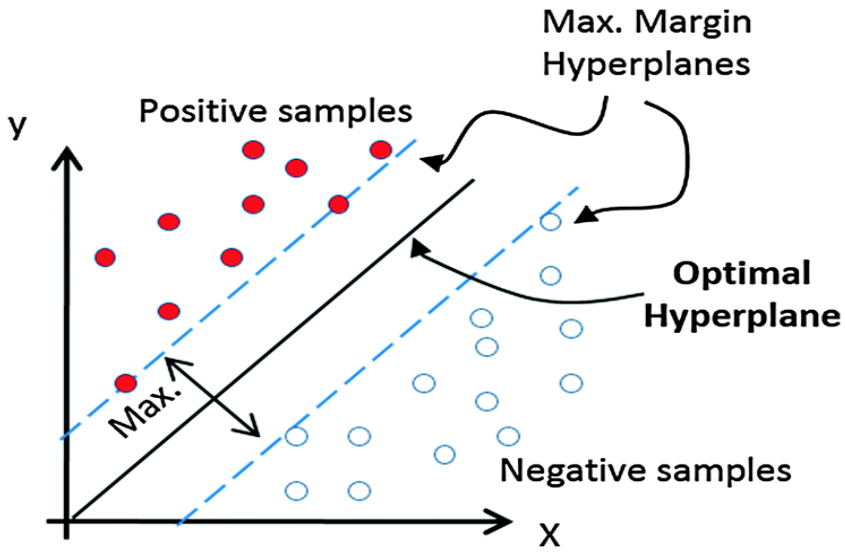


Figure 24. Example of a Support Vector Machine classifier separating examples with a suitable margin

The results assist the Multi-Objective Mapper to identify characteristics in the mappings that increase or decrease the deployment's efficiency, so the predictions can be included in the decision mechanism of the optimal deployment. Furthermore, the complexity offered by the amount of the data that is provided by the Monitoring Framework will guarantee for the design of a model that is able to discriminate with a high success rate the 'positive' mappings from the 'negative' ones, even in complex data situations like that shown in Figure 25.

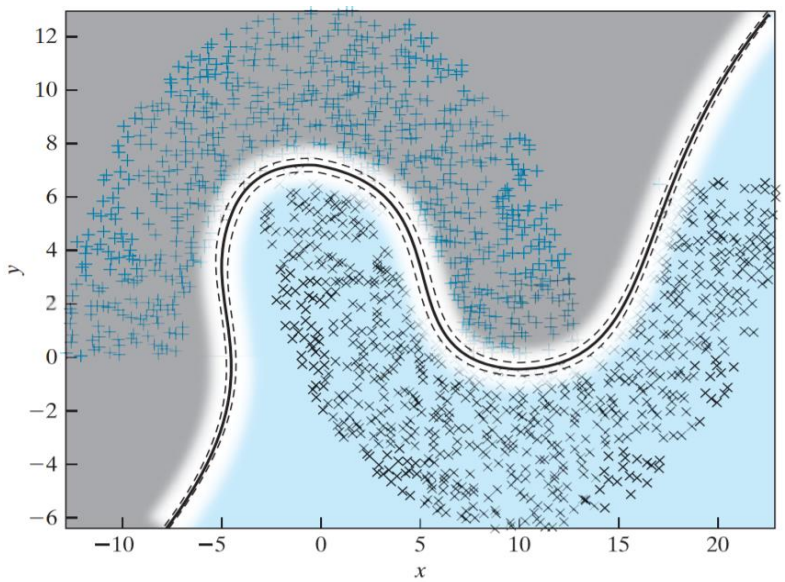


Figure 25 Example of a complex situation non-linearly separable, yet successfully distinguished

4.3.2 Failure events detection using previous metrics

The Monitoring Framework captures thousands of different events occurring during application execution regarding performance, utilization, energy costs etc. Depending on

the user's needs, different kinds of events can be defined as *target events*, and can be declared as such at run time. For example, if the user's interest resides on events where memory problems occur, the tool is responsible for predicting when these events' occurrence is possible.

During the data analysis, the tool uses a time window where it keeps track of all the events occurring at the window's time period and it correlates a set of other-type events with the occurrence of the target events. With the use of such predictions, the user is able not only to anticipate and prepare for such events, but he/she is also able to diagnose the reasons for the occurrence of the events. The tool's applications can be harvested by the PHANTOM system itself as well, since it can be used to identify, for example, problems with memory corruptions that could indicate misuse of the PHANTOM Programming Model.

The idea [9] behind this strategy is depicted in Figure 8 below:

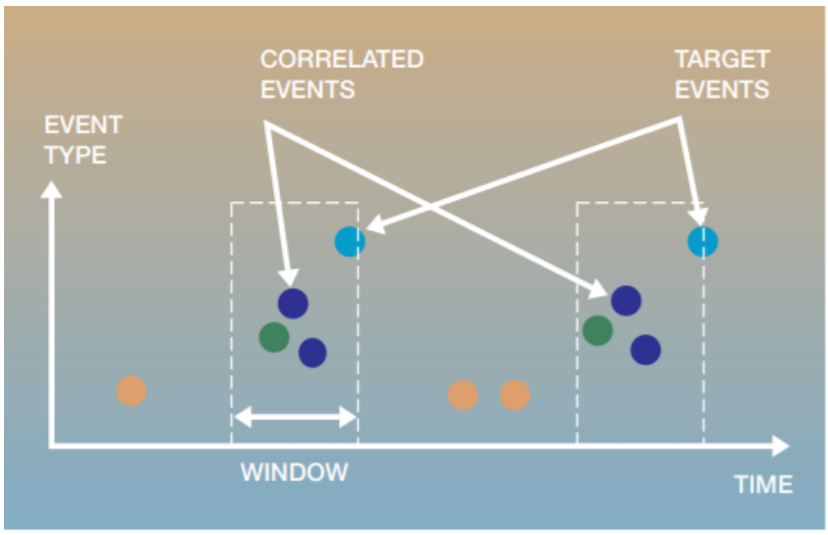


Figure 8. An example of a series of events along with the window that is used for the correlation of the target events with preceding events.

- ***Technical approach***

  The algorithm that is used takes the following steps:

  1. Uses associations to find frequent event-sets within the time windows preceding target events. In specific, all the events preceding the target events are recorded, forming different event-sets with high occurrence (a-priory) probability.
  2. Validates those event-sets against events outside the time windows considered in step 1. In this step, the tool uses a confidence metric of event-sets equal to the proportion of the event-sets' occurrences preceding target events over the total amount of the sets' occurrences over time.

$$confidence = \frac{event - set\ occurrences\ preceding\ target\ events}{total\ event - set\ occurrences}$$

The event-sets are considered validated for a high-confidence value and rejected for low-confidence values.

3. Builds a rule-based model for prediction. The tool locates the most specific and accurate rules first and eliminates the more general ones in order to predict the target events precisely.

4.3.3 Next steps

The integration process of the multi-dimensional optimization technologies is driven by the guidelines provided by WP5-" *Integration, use case implementation and validation*" as described in D5.1-"*Validation plan*".

Therefore, along with the integration process and validation tests on WP5 of all the PHANTOM components, the Monitoring Framework will be able to provide real data based on actual measurements on the use cases' performance. Based on this data, a series of evaluation tests will be executed in order to compare the efficiency of different algorithms and calibrate the different parameters that will enable the prediction models to perform with a high degree of accuracy.

4.4 INTERFACES

In this section, the different interfaces provided by the MF Server are described. The first one is the RESTful API which provides the functionalities for uploading to and retrieving data from the MF server. The functionalities provided by the API are not only used by the MF Client and the MF Library, and also by PHANTOM tools like MBT and Generic-MOM.

Additionally, the MF Server provides an interface based on web pages, which is more user-friendly and less prone to syntax errors, as the query scripts are implemented internally based on the fields of webpage forms. The interface provides, in addition to these forms, an integrated data visualization of data with the Open source tools of Kibana and Grafana.

4.4.1 MF-Server RESTful APIs

The MF-Server has to keep record of the applications or workflows (data such as the author, type of optimization, and the executable name), record of the executions or experiments (such as the name of the machine where it was executed and date of execution), as well as all the metrics collected during execution.

The following functionalities and services are supported by the PHANTOM MF-Server RESTful APIs (see Table 8), the purpose of which is to provide access to the data registered in the MF-Server:

- Registration of sampled metrics and timestamps from the MF Client.

- Provide means for users and tools to retrieve historical profiles of the monitored metrics. It is also possible to retrieve metrics filtered by timestamps, categorized by type (e.g. performance or power).

- Users and tools can obtain a fine-grained execution time of a sub-components of an application by specifying the unique generated execution ID.

- Since some configuration parameters of various devices, included in the heterogeneous infrastructure testbed, are stored in the MF server, it is possible to change these data via the RESTful APIs.

- Simple statistics and data analytics are fulfilled by the profile queries. It can be extended by providing user-defined queries, which we will refer as micro-queries to be run in the MF-server. It can help with debugging processes, and also for testing new queries before being integrated into future versions of the MF-Server.

Table 8: RESTful APIs of PHANTOM MF server

workflows (Applications registered in the MF-Server)		
/workflows	GET	Get all applications registered by the PHANTOM MF-Server
/workflows/:application_id	GET	Get specific application details, such as an id.
	PUT	Register a new application by the PHANTOM MF server

`Examples:`
`curl -XGET http://localhost:3033/v1/phantom_mf/workflows`
`Example considering an app named "demo" was registered.`
`curl -XGET http://localhost:3033/v1/phantom_mf/workflows/demo`		

Experiments (Individual executions of the registered applications)		
/experiments	GET	Get all available experiments
/experiments/:execution_id	GET	Get specific experiment details, such as an id.
`Example: curl -XGET http://localhost:3033/v1/phantom_mf/experiments`		
profiles		
/profiles/:application_id	GET	Get all historical metrics of a specific application
/profiles/:application_id/:task_id	GET	Get all historical metrics of a specific component
/profiles/:application_id/:task_id/:execution_id	GET	Get all historical metrics of a specific run of a specific component
`Example: curl -XGET http://localhost:3033/v1/phantom_mf/profiles/demo/pthread-example/AWU87m3quRkaX6JeceE9`		
runtime		
/runtime/:application_id/:task_id/:execution_id	GET	Get the execution time of a specific experiment
`Example: curl -XGET http://localhost:3033/v1/phantom_mf/runtime/demo/pthread-example/AWU87m3quRkaX6JeceE9`		
statistics		

/profiles/:application_id?metric=:metric_name	GET	Get statistics (e.g. average, minimum, maximum, etc.) of a metric value of a specific application
/profiles/:application_id/:task_id?metric=:metric_name	GET	Get statistics (e.g. average, minimum, maximum, etc.) of a metric value of a specific task
/profiles/:application_id/:task_id/:execution_id?metric=:metric_name	GET	Get statistics (e.g. average, minimum, maximum, etc.) of a metric value of a specific experiment
User-defined Micro Querying analytics		
/es_query_exec?QueryBody="\{\"query\":\{\" …. \"\}\}";	GET	Performs a micro query provided by the end user.

Some examples provided to ease the understanding of the syntax.

4.4.2 Execution Manager RESTful API

Table 9: RESTful APIs of PHANTOM Execution Manager

Queries on Experiments Data		
/ count_executions/:application_id/:task_id	GET	Count of the experiment_ids
Example: curl -s -XGET http://localhost:8700/count_executions?appid="demo"\&taskid="pthread-example"		
/list_executions/:application_id/: task_id	GET	List of the experiment_ids
Example: curl -s -XGET http://localhost:8700/list_executions?appid="demo"\&taskid="pthread-example"		
Collection of statistics		
/get_experiments_stats/:application_id /:task_id/:execution_id	GET	Query of stats
Example: curl -s http://localhost:8700/get_experiments_stats?appid="demo"\&taskid="pthread-example"\&execution="AWSTAGjEocAwx_GhqUgr"		
Statistics on the application components		
/get_component_timing/:application_id/: task_id	GET	Query for the duration of the application's components
Example: curl -s -XGET http://localhost:8700/get_component_timing?appid="demo"\&task_id="pthread-example"		
Query for User-defined Metrics		
/get_user_defined_metrics/:application_id/: task_id	GET	Full list of user metrics
Example: curl -XGET http://localhost:8700/get_user_defined_metrics?appid="demo"\&taskid="pthread-example"		

4.4.3 Some examples provided to ease the understanding of the syntax. User Interfaces

The ElasticSearch database and RESTful interface of the MF-server allow custom interfaces to be easily tailored to the user needs. In particular, HTML-javascript web-pages and graphic interfaces.

Interface based on HTML-javascript web-pages.

The web-pages use internally the RESTful API described in the previous section. Those pages allow the users to query the data just by providing their parameters in a form, This task is eased because the end users can use any device with a browser, without requirements on the operating system, without the need for end users to remember the syntax of the RESTful API. The next figure shows a screenshot of one of those pages.

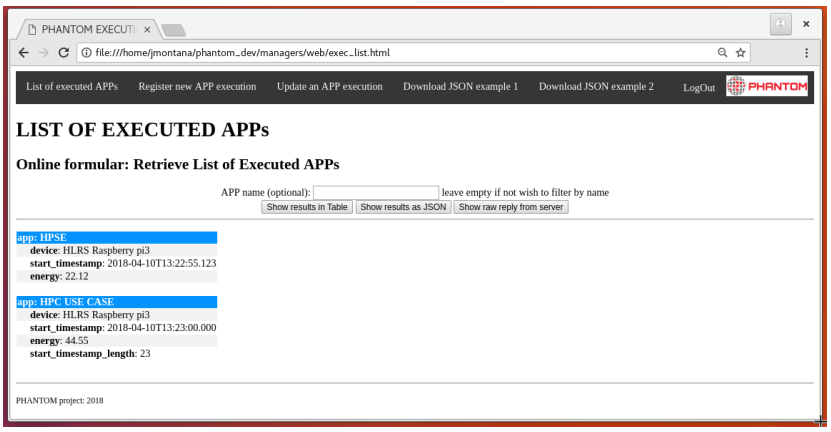


Figure 26: Screenshot of a query of executed user-applications.

Application-specific visualization with Grafana and Kibana.

The PHANTOM Monitoring Server allows several visualization back-ends (like Grafana and Kibana) to get access to the application-specific data, thus relieving the application from the need to exploit any additional visualization tools. The commodity tools use those visualization back-ends for infrastructure data only. We have considered both visualization tools because, given their similarities, it is not needed a significant effort to integrate both after integrating one of them.

Also for the application-level monitoring, the analytic aims to support the visualization, e.g. by applying filtering to the visualized data, extra- and interpolation of the inner or boundary values, accordingly.

Developed general visualization back-ends for both graphical (Grafana, Kibana) and command-line (textual representation) GUIs (a major contribution to task T4.3). In the previous projects, it was impossible to represent processed data. The purpose is to provide a default visualization of the metrics collected, which can be easily modified or create new ones only using the mouse.

The visualization in the previous projects plotted the time evolution of raw data from the database in simple Java GUI clients. However, the PHANTOM solution should support basic integration with the state-of-the-art visualization GUIs like Grafana or Kibana. The visualization back-ends can be used to plot data stored in the database. The next figures, Figure 27 and Figure 28, show screenshots of the visualization of metrics by Grafana and Kibana, respectively. The metrics shown in the figures were obtained from an execution of the example code in the *"Appendix 2. Example of Monitoring a Multi-thread Application."*.

The data stored in the MF-Server may need some processing when we wish to cross data from different types of data, such as video frames processed per watt, find relevant frames of data among the data stored, or other results from data analytics. Such processing of data from previous executions is done by the Execution Manager. Thus, the MF-Server will be easier to maintain in future implementations.

Notice that the MF-Server may have an intensive incoming traffic from the running applications. In such case, the separation of tasks among the MF-Server and the Execution Manager will be an advantage, because it will allow separation of their network traffic on different ports (may also different IPs). Such separation should avoid network bottlenecks in the service of the Execution Manager.

Figure 27: Screenshot of Visualization with Grafana of Disk-Write, Ram usage, CPU Load, and some User-Metrics. Data is generated by the example code in Appendix 2. Example of Monitoring a Multi-thread Application.

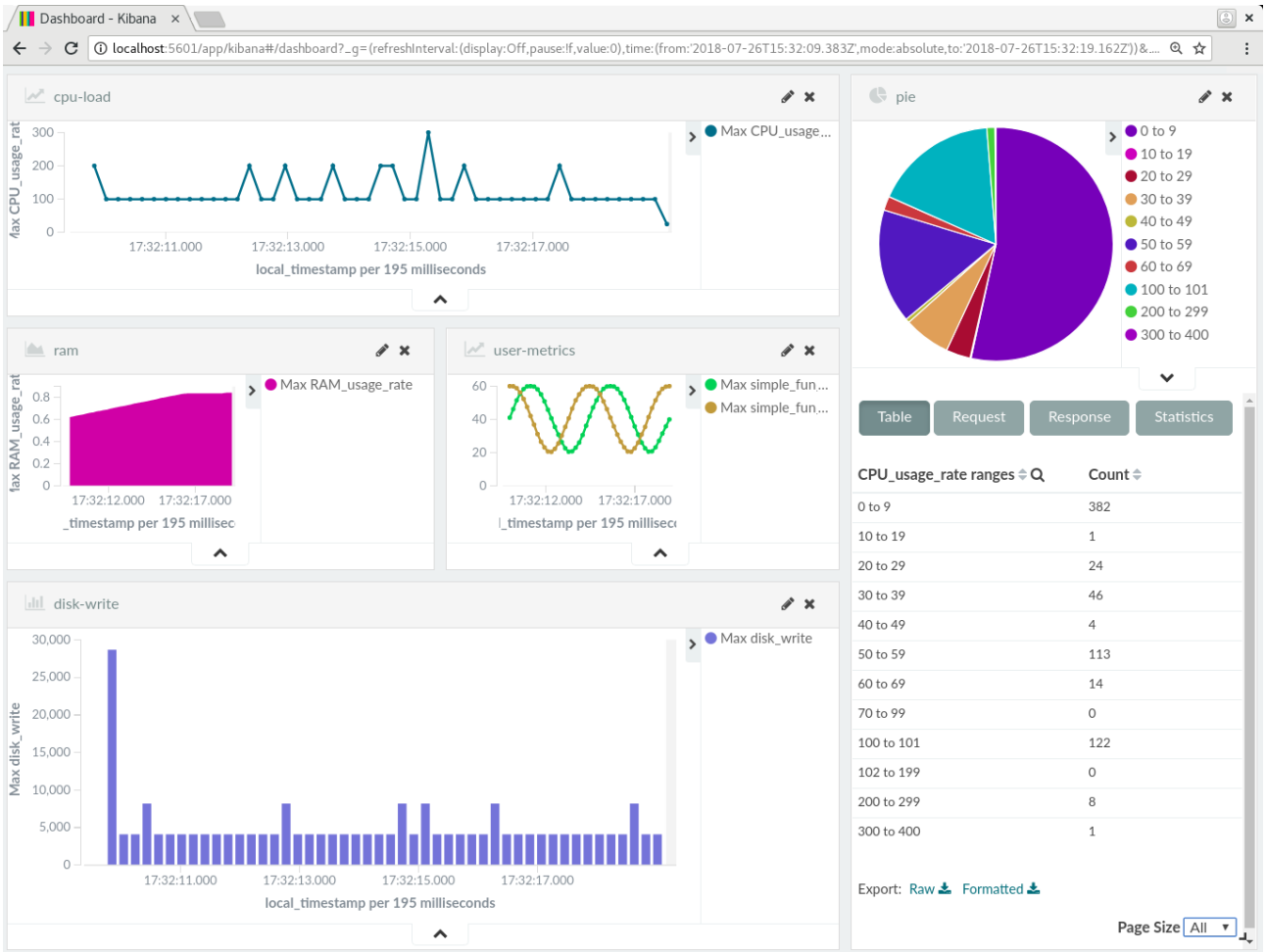


Figure 28: Screenshot of Visualization with Kibana of the same metrics represented in the previous figure (Disk-Write, Ram usage, CPU Load, and some User-Metrics).

4.5 INTEGRATION ASPECTS

4.5.1 Integration of PHANTOM tools with the MF-Server

The integration with the PHANTOM toolset is done through the interface implemented in a RESTful API. To facilitate this task, some simple examples are available in both Java and C in the GitHub repository.

The functionality of the RESTful API was described in section 4.4.1. Fundamentally, the MF-Server supports the following registered inputs: the workflows, the applications, and the collected metrics. It also supports to provide access to those registered metrics to the PHANTOM toolset (such as MOM and MBT) and the users.

The use of the RESTful API requires building HTML requests with some parameters, as shown in Table 8 and Table 9. However, we notice that building such requests cannot be easy to remember, and any error when writing the request will return an undesirable response. For this reason, we provide an interface intended for the end users, which can also be used by the partners developing PHANTOM tools, described in section 4.4.2.

That user interface consists of a set of HTML-javascript web pages that enable the use of the RESTful API, where the user only needs to fill a form with their parameters.

In addition to that, the collected metrics are in the correct format to be represented by visualization tools like Grafana and Kibana. Such visualization tools allow users to easily represent them and modify the registered representations only using the mouse.

4.5.2 Integration of the MF-Server with the Security server.

The integration of the MF-Server with the security server allows limiting access to data only to those that are authorized.

In order to provide such access control, the users have to be identified, and the data must have associated with different access domains. To achieve this, an access *domain* is assigned each time an experiment[3] is created[4]. The identification of the users is provided by a unique key. The access is authorized or denied by the NGAC depending on the provided key, which we refer as authorization *token*.

NGAC decides whether a user has read or write permission in each *domain* based on an *access policy definition*. The *access policy definition* contains the list of registered users, the list of domains, and r/w permissions of those users in those domains.

Below are shown examples of the generation of tokens, and their use when requesting access to data from a particular experiment. Appendix 4 "Examples of Security Policy Configuration Files" shows an example of an *access policy definition*.

Example of *token request* providing a user id and a password (replace the port by the corresponding port, the default port is 8700 and 3033 for the Execution manager and the MF-server, respectively):

```
curl -s -H "Content-Type: text/plain" -XGET http://localhost:8700/login?email="hpcjmont@hlrs.de"\&pw="XXXXX" --output token.txt;
mytoken=`cat token.txt;`;
```

Example of *data request* with authorization token:

```
curl -s -H "Authorization: OAuth ${mytoken}" -H "Content-Type: multipart/form-data" -XGET http://localhost:8700/get component timing?appid="demo"\&task id="pthread-example"\&execution="AWSTAGjEocAwx_GhqUgr";
```

[3] We refer as experiment to each execution of a registered application in the MF-Server.
[4] We consider defining the access permissions on the experiment level, because it allows deciding which monitored executions be shared with different users, and because it seems not efficient to define access permissions for the entries on the measurements level that can be numbered in the order of millions.

Result for unauthorized access:

```
403: Access denied
```

Example of an authorized result:


```
{
  "total": 2,
  "max_score": 1,
  "hits": [
    {
      "_index": "demo_pthread-example",
      "_type": "AWSjr040Pct1kMuA6n1E",
      "_id": "AWSjr1G6Pct1kMuA6n4B",
      "_score": 1,
      "_source": {
        "TaskID": "pthread-example",
        "type": "user_defined",
        "host": "node01",
        "local_timestamp": "2018-07-16T17:22:12.385",
        "component_duration": "2084652442",
        "runid": "2018-07-16T17:22:08.921",
        "component_start": "31947963672078",
        "component_end": "31950048324520",
        "server_timestamp": "2018-07-16T17:22:13.304"
      }
    },
    {
      "_index": "demo_pthread-example",
      "_type": "AWSjr040Pct1kMuA6n1E",
      "_id": "AWSjr1G6Pct1kMuA6n4C",
      "_score": 1,
      "_source": {
        "TaskID": "pthread-example",
        "type": "user_defined",
        "host": "node01",
        "local_timestamp": "2018-07-16T17:22:12.386",
        "component_duration": "2027679398",
        "runid": "2018-07-16T17:22:08.921",
        "component_start": "31948030973731",
        "component_end": "31950058653129",
        "server_timestamp": "2018-07-16T17:22:13.304"
      }
    }
  ]
}
```


4.6 SUMMARY OF INNOVATIONS BEYOND SOTA

The following major innovations are identified for the Monitoring Server:

- **Monitoring "data lane" customized "in data" analytics.** The monitoring server improves the efficiency by allowing to perform the data analytics at the place where the data is stored. This is more efficient than sending a large amount of data to be processed in another location.

- **The split of gathering, storing and analysis technologies.** The MF-Server collects the data and stores it in an ElasticSearch database. The MF-Server, as playing the role of the interface, can provide a consistency filter of the collected data, and filter possible errors. The analysis of previous executions and generation of statistics is done by the Execution Manager, which can prepare such results to be requested as soon as the metrics are available, which may improve the response time.

- **Open interfaces fostering collaboration with the other tools**

  - **Analysis easily extended to the users' needs:** The analysis functionality can be easily extended to the users' needs. Both MF-Server and Execution Manager are open-source, and both support user-defined *micro-queries* directly provided by the users.

  - **Visualization backends:** Other tools' developers can access all the stored data in the database, which can support to debug their tools and analize the behaviour of the user applications.

- **High scalable on distributed platforms:** The PHANTOM servers, as well as the ElasticSearch database, can run on distributed platforms supporting high scalability of the system.

4.6.1 Background technologies utilised in the development

Among a broad set of available open-source tools dealing with infrastructure monitoring (e.g. Zabbix, Nagios), and application-level optimization (e.g. Paraver, Vampir), we were unable to identify any technology that would fulfill the user's requirement to the Monitoring Framework – i.e. allow the collection and processing of the application-specific data.

Therefore, the MF-Server extensions to support the user-defined metrics – the Monitoring Library – were largely developed from scratch (with the reuse of the design outcomes of the EXCESS[5] project). The elaborated API syntax follows the one used in the well-established tools for parallel applications profiling like Extrae[6] and Vampir-Trace[7] and is tightly integrated with the Monitoring Client functionality (cf. D4.2).

4.6.2 Summary of new technologies/extensions developed

The users can analyse the changes in the collected metrics across multiple executions of the application. Also for the MBT tools, the user-defined metrics can give important hints on the properties that should be considered during the modelling.

[5] http://www.excess-project.eu
[6] https://tools.bsc.es/extrae
[7] https://tu-dresden.de/zih/forschung/projekte/vampirtrace

After the execution, the users can perform the analytics on their customized metrics in the same way as they would do it for any default metric, using the macro- or micro-querying functionality of the Monitoring Server (see in D4.2).

Integration with command-line visualization back-ends. Unlike the commodity and HPC systems, the embedded and low-power devices cannot perform visualization with compute-intensive tools like Grafana and Kibana. For this, the RESTful API enables the visualization through the command line (in a textual form).

4.6.3 Source release and GIT repositories

The source code of PHANTOM monitoring framework is released using GitHub, which is an open-source code management system. In order to facilitate the development process, we follow the Git workflow paradigm, which describes a common branching model. The source code is split into two repositories on GitHub, namely the "phantom_monitoring_server", which can be accessed via the following links:

https://github.com/PHANTOM-Platform/Monitoring/tree/master/Monitoring_server

5. THE PHANTOM SECURITY FRAMEWORK

Deliverable D1.2 identified two loci of effort within the PHANTOM security development:

- A low-level solution to ensure integrity of component network execution on the heterogeneous PHANTOM platform based on isolation and information flow control (IIFC) inspired by the MILS approach
- A high-level solution for unified and flexible access control across enterprise-scale systems that span heterogeneous operating environments using the Next Generation Access Control standard

Deliverable D1.3 described the design of the PHANTOM security methodology and a further refinement of the approach.

5.1 REQUIREMENTS

The following subsections discuss the status of the requirements, which consist of two sets:

- The initial set of requirements, imposed by the use case providers at the beginning of the project and specified in Deliverable D1.1;
- Additional requirements that were obtained from the feedback of use case providers, done after the release of preliminary version of PHANTOM tools by M18.

5.1.1 Initial Requirements

Table 10: Initial requirement to Security Framework

ID	Requirement	Priority	Current Status
U64	Remote target platforms should be able to be secured against eavesdropping through interfaces with external infrastructures for trust/authentication	SHOULD	Achieved through operating environment mechanisms for secure communication.
U65	Data obtained through the run-time monitoring should be able to be secured against eavesdropping / unwanted access	SHOULD	Monitoring data stored in the MF-Server which is under NGAC based access control
U66	PHANTOM should support means for tasks isolation and information flow control policy	SHOULD	PHANTOM uses bespoke mechanisms for FPGA task isolation and OS provided mechanisms for GPU/CPU task isolation.

U67	PHANTOM should be able to support HPC/Cloud security mechanisms to protect data and control data access	SHOULD	May be achieved through typical third-party security mechanisms. This does not conflict with PHANTOM.
U68	PHANTOM shall be able to guarantee data integrity when applications are mapped onto heterogeneous targets	SHALL	The combination of the execution integrity measures for FPGAs, which can be employed when desired, along with the OS provided isolation of GPU/CPU processes, and the NGAC controls on the Repository meets this requirement.

5.1.2 Preliminary Feedback from M18 Results

The use case partner GMV has already used the declarative policy language and the lightweight 'ngac' policy tool to model and test an appropriate security policy for the Surveillance Use Case. They were able to do this quickly using only the provided documentation and without support from the tool implementers.

The developers of the PHANTOM Repository (HLRS) were able to integrate with the Policy Server after a short introduction to the concept and the server API.

Example polices resulting from these two uses by partners are presented in Appendix 4.

5.2 DESIGN SPECIFICATIONS

Security framework design issues were previously introduced in Section 3.7 of D1.3. Here we discuss more recent refinements.

5.2.1 Design of Execution Integrity for Component Network

The PHANTOM programming model provides the ability to construct an application that is a composition of communicating, cooperating components referred to as a *component network*. The component network is a modular construction of application or PHANTOM components (operational components) that enables component reuse and reasoning about the operation of the component network based on the known functional attributes of the components and the manner in which they are composed in the component network.

When reasoning about the composition of components in a component network a critical *assumption* is implicitly made: that the execution environment guarantees that the abstraction of the component network is faithfully refined in its realization. Specifically, that the distinctness of components represented in the component network, the unique source and destination of each inter-component communication connection, and the absence of interference among components, or upon components and their

communications by other parts of the system, are preserved in the execution environment provided by the PHANTOM platform.

The properties of the platform that provide the basis for the component network assumption are: *isolation* and *information flow control* (IFC). That is, the platform as a whole, and each node (hardware processing element along with its control firmware and software), must guarantee that each exported resource (process, memory, etc.) is free from undefined external interference (isolation), that only defined (by the component network) inter-component interference may occur through communication mechanisms provided by the platform, and that such communications are free from interference from other sources (IFC).

The ability of the PHANTOM platform to provide isolation and IFC to a component network is based on the ability of each of its nodes to provide isolation and IFC for the resources it exports that are involved in the component network. Hence, we provide an account for the manner in which each type of node in the heterogeneous PHANTOM platform provides isolation and IFC for the resources it exports. Together, we refer to the combined effect of these properties of isolation and IFC of the separate processing elements as *execution integrity* of the component network.

5.2.2 Execution Integrity of CPU Processes

Generally, PHANTOM software components run as, or are supported by CPU processes. These processes are created by the kernel of a general-purpose operating system (OS) such as Linux, or a real-time operating system in certain applications. The isolation of software components relies on standard Linux kernel process isolation. Each process is run as an ordinary user with limited privileges to avoid any potential interference. Some system processes are run as special system user having greater privileges.

Part of the service provided by the OS is the management of the primitive resources that are shared among the processes being run by the OS, including main memory, CPU cycles, and hardware devices that are integrated with the CPU such as the system clock or a GPU. Other hardware devices, considered add-on peripherals, such as mass storage devices and network devices, are controlled by driver software and associated subsystems such as file systems and network stacks. These trusted software systems in turn provide separation among the abstract resources associated with the primitive resources of these devices as they are being used by different users and processes acting on behalf of those users.

In the case of CPU processes the OS and the hardware on which it runs are *trusted components* of the platform. From the standpoint of (application) component network execution integrity these platform (foundational) components are trusted to correctly provide isolation and IFC for (application) software components running in CPU processes.

5.2.3 Execution Integrity of GPU Processes

Application component code running on the Linux kernel is afforded isolation and interference protection by the standard mechanisms in the kernel. GPU drivers attempt

to provide similar mechanisms, by grouping memory into contexts. A context is owned by an application process and may only be read by that process. Memory on the device is allocated to a context by the GPU drivers, and it cannot be accessed by other processes unless explicitly declared as shared memory. For most use cases this is sufficient.

GPU processes are potentially susceptible to memory leakage and object reuse problems. Unfortunately, these problems make it difficult to reason about security in the GPU domain. First, driver code to interact with the device is both proprietary and secret. This means that we cannot perform the security audits that we can with open source code, and we cannot edit the code to improve it with attack mitigation techniques. Second, the focus of GPU drivers is on performance first, and security second (or not at all). Unlike a standard OS, most GPU implementations do not zero memory that is allocated to a process, meaning that it may contain data from a previous user. Even worse, GPUs make it difficult to erase the entirety of your own memory, as one can do on a standard OS process, because of the existence of constant memory and other optimisation-focussed limitations. There are many live attacks against current GPU implementations, and little that can be done by anyone other than the manufacturer to prevent them. If high security is required then it is recommended that one of the following options be taken:

1. Avoid the use of the GPU from the process to be secured from exfiltration. FPGAs can be hardened with confidence so should be considered a better option.

2. If the GPU is to be used, ensure that the only process using it is the one that is being secured, and prevent all other processes from accessing the GPU.

5.2.4 Execution Integrity of FPGA Processes

In most systems, hardware is trusted with full system access. However, in the PHANTOM environment FPGA hardware designs come from the user and cannot be trusted, so it is necessary to ensure that hardware components can access only the memory and devices associated with their software part, and nothing else. If any component mapped to the FPGA infrastructure requires security isolation then additional capabilities are automatically added to the infrastructure to ensure that malicious IP cores cannot affect system integrity, and that cores handling sensitive data are protected from observation by other cores. We present here a summary of the approach taken; details are provided in deliverable D4.3.

A PHANTOM FPGA component mapped to an FPGA consists of a hardware part and a software part. The software part exists as a standard Linux process, and the hardware part is an IP core implemented on the reconfigurable logic fabric of the FPGA. This provides a unique security challenge, because it is essential that the software part of a component can communicate with its hardware part and vice versa, but it must not be possible for the software to communicate with the hardware from a different component.

The isolation of software components relies on standard Linux kernel process isolation. As noted above, each process is run as an ordinary user with limited privileges to avoid potential interference. Hardware components are controlled from the ARM cores using memory-mapped registers in the CPU's standard address space, as well as including a portion of shared memory for each component that both the hardware and software can access. Hardware components are mapped into the Linux kernel's UIO device space, with a device node being created for each device. The permissions of this node are then set to ensure that only the correct corresponding software processes can access them. This is automatically implemented using the Linux device manager udev, and enforced by the kernel.

To ensure that hardware components can only access the memory to which they are permitted, a minimal memory management unit (MMU) is added to the memory connection of each hardware device. Figure 29 depicts the AXI SmartConnect MMU used to restrict hardware access to memory on FPGA devices. The MMU is preprogrammed by the system infrastructure and cannot be changed by the system at runtime, thereby isolating components into their own memory spaces.

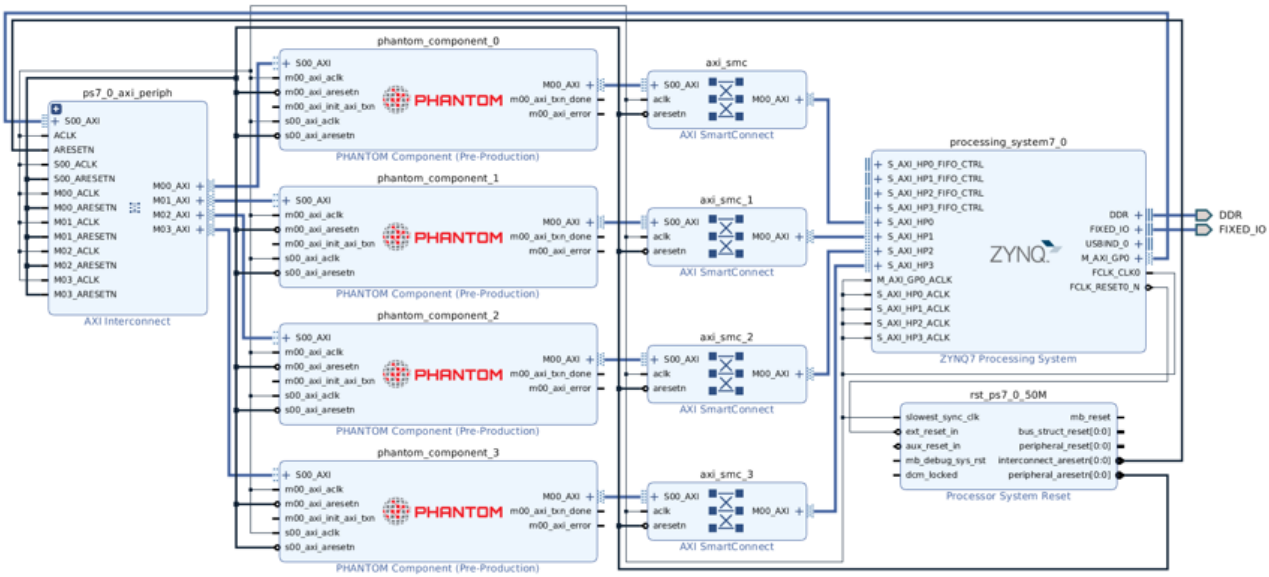


Figure 29: AXI SmartConnect MMU restricts memory access on FPGA devices

The AXI bus is not a multi-drop bus, but rather a set of point-to-point connections. Reads and writes made by one IP core are not visible to other cores. The structure and address ranges of the bus are all defined statically at architecture-build time. Each bus endpoint has a range of addresses that it responds to, so that at runtime a read of any given address will always go to a specific location and no others. These ranges cannot overlap or the bus fails. Endpoints filter out transactions for addresses they are not responsible for, and this filtering is done statically by the constructed architecture, not by any user-accessible logic in the IP cores. Thus, the solution provides full protection.

5.2.5 Next Generation Access Control (NGAC) Functional Architecture

The NGAC functional architecture (NGAC FA) [10] is illustrated in Figure 30, where PEP = Policy Enforcement Point, RAP = Resource Access Point, PDP = Policy Decision Point, EPP = Event Processing Point, PAP = Policy Access/Administration Point, and PIP = Policy Information Point. Requests to access *protected resources* come from applications that are implemented to access an API provided by the PEP. The PEP consults a PDP to determine whether a particular request should be performed, and if the PDP responds in the affirmative, the PEP will proceed to access the resource through the RAP and return the result to the requesting application. If an optional EPP is implemented, the PEP will also notify the EPP of its actions. A PDP will access the stored policy that resides in the PIP through a PAP. The PAP can also be used to initialize or modify the policy in the PIP. These functional components may be implemented separately or monolithically, but in any event are part of the *trusted computing base*.

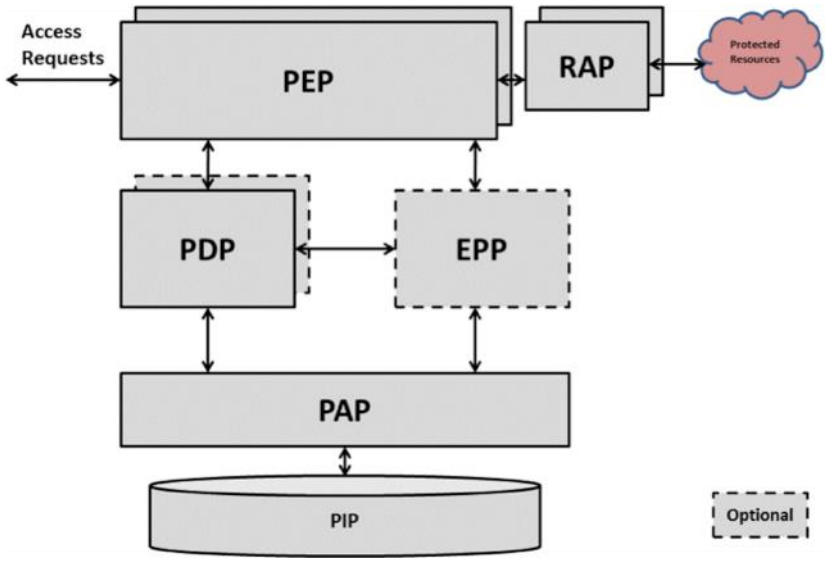


Figure 30: NGAC functional architecture

5.2.6 NGAC Investigation and Architectural Design Approach

Our interpretation of the generic NGAC FA, see Figure 31Figure 31, is used as the template for our design and implementation of NGAC. It is distinguished from the generic NGAC-FA by identifying a resource access path (orange), an access control path (green), and several open interfaces (black bars along these two paths). In our interpretation the Policy Server includes the PDP, PAP and PIP. Also, in our interpretation, although the PEP and the RAP are still trusted components they are placed into the domain of the application developer. The developer may define the policy enforcement interface and the resource access interface as may be convenient for the application. The developer is required to follow this architectural pattern and to keep these modules very simple so that they may easily be verified by inspection, if not by more rigorous analysis. The PEP(s) developed by the developer use the standardised Policy Query Interface defined by the Policy Server to access the PDP.

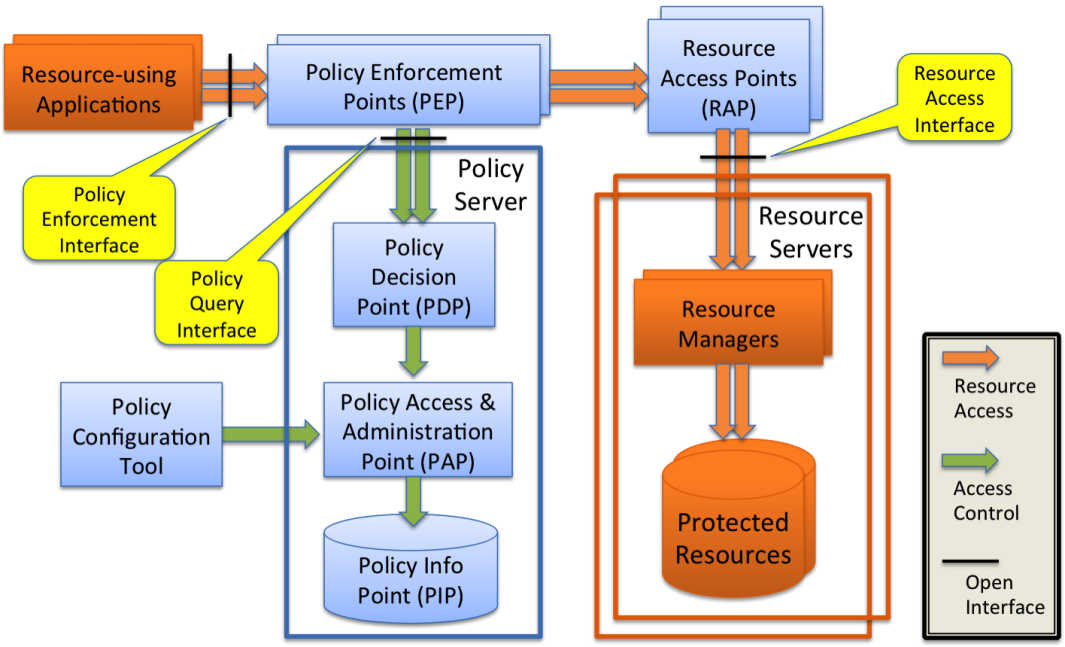


Figure 31: Refinement of the NGAC functional architecture for PHANTOM

This approach, based on our interpretation and factoring of the NGAC FA, is feasible due to our development of a more lightweight and portable Policy Server. We describe the events leading to our chosen design approach.

The previously available NGAC reference implementations, initially considered for PHANTOM, are very heavyweight, requiring a specific and cumbersome operating environment and supporting extension packages. Only the enterprise-scale applications in PHANTOM could be expected to be capable of supporting the early NGAC reference implementations, ignoring the other difficulties they posed, such as the monolithic approach to implementation of the NGAC FA and the difficulty of modification, extension, build and maintenance.

As part of the PHANTOM security effort we studied the feasibility of re-implementing the heavyweight reference implementation more portably, or alternatively implementing a more lightweight and portable approach. As a result of this investigation and some exploratory implementation, we concluded that a lightweight and portable server would be achievable within the Project, and thus we proceeded with that implementation.

5.2.7 Design of the 'ngac' Policy Tool

Though the NGAC standard defines a policy specification framework and the reference implementations provided a more or less monolithic implementation, the early NGAC reference implementations did not provide a practical way of developing and testing policies within the NGAC policy framework. The centralized server maintained a sole, system-wide policy instance that could be used for access control, and it could not be

run on ordinary workstations. Furthermore, the creation of a policy had to be done using one imperative command at a time using an admin tool that communicated the commands using an internal protocol to the server, which executed them on the current internal policy database. This made it impossible to conveniently and safely develop and test new policies, since the single server was shared with production activity.

To fill this gap, we created a declarative policy language and a desktop tool to read this language and to build an internal policy representation from which it could then compute policy decisions. The design of the language is described in the following section.

The formulation of policies in the NGAC policy framework can be represented as a directed graph, formed by users, objects, attributes, and relations among these. The access results are arrived at by computations over these graphs. We chose to implement the tool in Prolog, which is well suited to this problem for several reasons, among which are the natural way such entities and relations can be represented in Prolog, its ease of expressing and compactness of graph computations (in large part declaratively), the ease of doing linguistic and symbolic manipulations, its useful libraries, and its suitability for prototyping.

The tool, which we refer to as the 'ngac' policy tool, is structured as a collection of functions and supporting functions that can be invoked by the 'ngac' command interpreter. The functions are grouped according to: definition and processing of the declarative policy language, input and output of policy representations, policy representation conversions, computations over the internal policy representation, an extensible self-test framework with built-in test cases, and a few utility functions. The command interpreter can also run 'ngac' command scripts that may be predefined in the tool or provided in files by the user.

5.2.8 Design of NGAC Declarative Policy Specification Language

Based on the NGAC policy framework defined in the NGAC standard we have designed a declarative policy specification of the form:

policy(<policy name>, <policy root>, <policy elements>).

where,

<policy name> is an identifier for the policy definition

<policy root> is an identifier for the policy class defined by this definition

<policy elements> is a list [*<element>, ... , <element>*]

where each *<element>* is one of:

user(<user identifier>)

user_attribute(<user attribute identifier>)

object(<object identifier>, <object class identifier>, <inh>, <host name>, <path name>, <base node type>, <base node name>)

object_attribute(<object attribute identifier>)

policy_class(<policy class identifier>)

assign(<entity identifier>, <entity identifier>)

associate(<user attribute id>, <operations>, <object attribute id>)

where *<operations>* is a list:

[*<operation identifier>, ... , <operation identifier>*]

connector('PM')

The initial character of all identifiers must be a lower-case letter or the identifier must be quoted with single quotes, e.g. *smith* or *'Smith'* (identifiers are case sensitive so these examples are distinct). Quoting of an identifier that starts with a lower-case letter is optional, e.g. *smith* and *'smith'* are not distinct.

Additionally:

< inh > can be **yes** or **no**. It is not currently used, as are also *<base node type>* and *<base node name>*.

< host name> contains the name of the host where the corresponding file system object resides.

< path name> is the complete path name of the corresponding file system object.

5.2.9 Design of NGAC Policy Server

The Policy Server depicted in Figure 31 must provide a Policy Decision Point (PDP), a Policy Access Point (PAP), and a Policy Information Point. The policies are stored as ordinary files containing policy specifications in the NGAC declarative policy language. This currently comprises the persistent policy store, which can be maintained in a directory that cannot be written by users not privileged to set policy. Policies may be created with any text editor, tested with the 'ngac' tool, and placed into the policy store when they are ready for use. To make policy decisions at runtime, however, policies need to be read-in to a store that is internal to the server in a form that is amenable to computation. This store is the runtime PIP.

The Policy Server can utilize the policy input, policy storage, and policy computation algorithms of the 'ngac' policy tool. An HTTP-based interface is added to provide a Policy Query Interface that can be called by multiple Policy Enforcement Points (PEPs).

5.3 IMPLEMENTATION DETAILS

5.3.1 Implementation of Execution Integrity for Component Networks

Component network execution integrity is achieved by the architectural design of the PHANTOM execution environment and by confirmation that the assumptions on the platform that are necessary for execution integrity are met.

As discussed in the foregoing design section, the implementation of execution integrity involves diverse existing mechanisms in the operating environment as well as special components provided for FPGAs. In PHANTOM, though FPGAs are used within a framework in which the FPGA IP core is supported by OS processes and protections within the OS's scope of control, OS protections do not extend to execution on the FPGA itself. Additional countermeasures are needed to maintain separation among cores sharing the same FPGA. Section 5.2 of this document, and document D4.3, describe the AXI SmartConnect MMU used to restrict hardware access to memory on FPGA devices.

The design section also discusses GPU devices, which in PHANTOM are used as a coprocessor to a CPU. The OS running on the CPU treats GPU state as part of the hardware state of processes that use the GPU, which it saves and restores during process switches.

The primary vehicle for the implementation of protection of execution integrity is therefore the host operating system of the involved CPUs, in our case Linux. GPUs are more closely bound to the CPU, and are treated as part of the state of a CPU process. For FPGAs the hardware interfaces between the CPU to the FPGA may be manipulated directly by the associated CPU process, and thus handled by the OS, but the FPGA hardware and its configuration are external to the CPU process. Hence the need for FPGA-specific mechanisms that we have described in the design and the implementation of which are further described in D4.3.

5.3.2 Implementation of the 'ngac' Policy Tool

The 'ngac' policy tool presents a command line interface that offers a defined set of commands. The command interpreter also offers selective tracing of commands for development and debugging of the tool itself. The 'ngac' command interpreter is easily extensible for new commands and this ability has been frequently used during its implementation. The syntax and simple semantic checking of commands are achieved declaratively and the addition of a new command is straightforward. There are two levels of commands: a restricted set for ordinary users and an expanded set that includes commands that are primarily of use to the tool developers. This capability was used heavily during development as commands have come and gone. The available command set is determined by a parameter and it may be changed by a command. The current version of the code has had much removed that had been superseded by later development.

When the tool is started it displays a banner and the command prompt “ngac>”. Among the current and most useful commands are:

- `version.`
  Display the current version number.

- `versions.`
  Display past versions with description and current version.

- `help.`
  List the commands available in the current mode.

- `help(` <command name> `).`
  Give a synopsis of the named command.

- `import_policy(` <policy file> `).`
  Import a declarative policy file and make it the current policy.

- `newpol(` <policy name> `).`
  Declare a new current policy.

- `access(` <policy name>`,` <permission triple> `).`
  Check whether a permission triple is a derived privilege of the policy.

- `proc(` <procedure name> `[, verbose]).`
  Run the named command procedure, optionally verbose.

- `proc(` <procedure name> `[, step]).`
  Run the named command procedure, optionally pausing after each command.

- `script(` <file name> `[, verbose]).`
  Run the named command file, optionally verbose.

- `server(` <port > `).`
  Start the server on the given port number.

- `script(` <file name> `[, step]).`
  Run the named command file, optionally pausing after each command.

- `echo(` <string > `).`
  Print the argument string, useful in command procedures.

- `nl.`
  Print a newline, useful in command procedures.

- `regtest.`
  Run built-in regression tests.

- `halt.`
  Exit the tool.

There are other commands defined in the `command` module, but some are esoteric and some obsolete.

The Prolog modules of the implementation of the 'ngac' tool are (the module names are the file names without the ".pl"):

- `ngac.pl` – top level module of 'ngac' policy tool; entry point and initialisation
- `param.pl` – global parameters
- `command.pl` – command interpreter and definition of the 'ngac' commands
- `common.pl` – simple predicates that may be used anywhere
- `pio.pl` – input / output of various policy representations
- `policies.pl` – example policies used for built-in self-test
- `test.pl` – testing framework for self-test and regression tests
- `procs.pl` – stored built-in 'ngac' command procedures
- `pmcmd.pl` – PM RI command representations and conversions
- `spld.pl` – security policy language definitions

In addition, a test module for any of the main modules may appear in the `TESTS` directory.

The tool can be easily extended in several ways.

- Commands can be added by modifying the `command` module to add a `syntax`, `semantics` (optional), `help`, and `do` clause for the new command. A `syntax` clause must be added for the command. This clause declares the command name and parameters, and what mode the command belongs to, admin or advanced. Admin commands are available in admin mode, but also accessible in the advanced mode but not vice versa.
- The self-test framework is implemented in the `test` module. Tests for specific new modules can be added in the `TEST` subdirectory. An example of a test definition file for the `spld` module is implemented in `TEST/spld_test.pl`.
- New predefined 'ngac' command procedures can be added to the `procs` module. A `proc` clause is added for each new procedure to be defined. There are examples in the `procs.pl` file.

Global parameters are set in the `param` module.

5.3.3 Implementation of NGAC Policy Server

The 'ngac-server' presents a RESTful API referred to as the Policy Query Interface. The server implementation is actually an addition to the 'ngac' tool implementation. It is currently started as a command of the 'ngac' tool. (See the `syntax`, `semantics`, `help` and `do` clauses for the `server` command in the `command` module.) The source can easily be configured to make a version that allows the server to be executed directly.

The Policy Server functionality actually includes the Policy Query Interface API, the PDP and the PAP/PIP as depicted in Figure 31.

The APIs currently implemented for the Policy Query Interface are:

pqapi/access – test for access permission under current policy
Parameters
- user = <user identifier>
- ar = <access right>
- object = <object identifier>

Returns
- "permit" or "deny"

pqapi/getpol – get current policy being used for policy queries
Parameters
- none

Returns
- <policy identifier> or "failure"

pqapi/setpol – set current policy to be used for policy queries
Parameters
- policy = <policy identifier>

Returns
- "success" or "failure"

pqapi/getobjectinfo – get object metadata (experimental for PEP to get object metadata stored in the object clauses of the policy)
Parameters
- object = <object identifier>

Returns
- "object=<obj id>,oclass=<obj class>,inh=<t/f>,host=<host>, path=<path>,basetype=<btype>,basename=bname>"

A couple more RESTful APIs may need to be added to get the best results when used with the PHANTOM Repository. For example, an API to cause a policy file to be

loaded into the server, and an API to store a session token for token- rather than user-based `access` queries.

The Prolog module for the implementation of the Policy Server is in the file `server.pl`. It implements the simple API described above but also contains stubs for some other APIs corresponding to a recent web-based reference implementation.

5.3.4 Installing and Running

The 'ngac' policy tool and server are implemented in Prolog and require the SWI Prolog environment to run. The software is provided as a set of Prolog source files and/or as an "executable" that has the Prolog runtime environment already bundled in. The executable for the 'ngac' policy tool is made by the shell script mkngac, located with the source files, that must be run in an environment that has SWI Prolog installed.

SWI Prolog is available for several operating environments, including Mac, Windows, and Linux. See http://www.swi-prolog.org.

The current version of the software consists of a directory including source files and example files and several sub-directories.

- If a ready made executable 'ngac' has been provided it may be executed directly from a command shell prompt. If you do this skip down to "Now you should see …" below.
- Otherwise, in the source directory ngac-tool-lwserver-2018-05 start SWI-Prolog from a command shell prompt using the name of the SWI-Prolog executable (usually 'swipl', 'swi-pl', or something similar, depending on how it was installed).
- After printing a short banner SWI-Prolog will display its prompt "?- ".
- At the Prolog prompt enter "[ngac]." (not the quotes)
- Prolog will compile the code and print "true."
- Execute the code by entering at the Prolog prompt "ngac."
- Now you should see the 'ngac' prompt "ngac> '
- The 'ngac' commands are now available. Entering the command "help" will list the available commands in the current mode.

The 'ngac' tool has some self-tests built in. These should be run to ensure that everything is working correctly. The self-tests can be run by starting 'ngac' normally and entering at the 'ngac' prompt the command "selftest."

The Policy Server is included in the same source directory.

The ngac Policy Server is currently started from the 'ngac' policy tool. After starting 'ngac' it offers the prompt "ngac>". If you want to load any policy files, do it now with the 'ngac' command "import(policy(PolicyFileName)).", where PolicyFileName is the name of a .pl file relative to the execution directory.

When you have the desired policies loaded, start the server from the 'ngac' tool using the command "server(PortNumber).", where PortNumber is an unused TCP port. The server will be started and will be listening to that port for calls to its RESTful API. There is a shell script of curl commands included with the source (servercurltest.sh). This script can be run to send a sequence of requests to the server to test for known correct answers for the built-in test policies.

5.4 INTEGRATION ASPECTS

The integration of the PHANTOM Repository Server with the PHANTOM Security Framework is based on the definition of the **access domain** in metadata of the uploaded contents to the Repository and the **user-id**. The Figure 32 shows an example of the metadata of two uploaded files to the Repository.

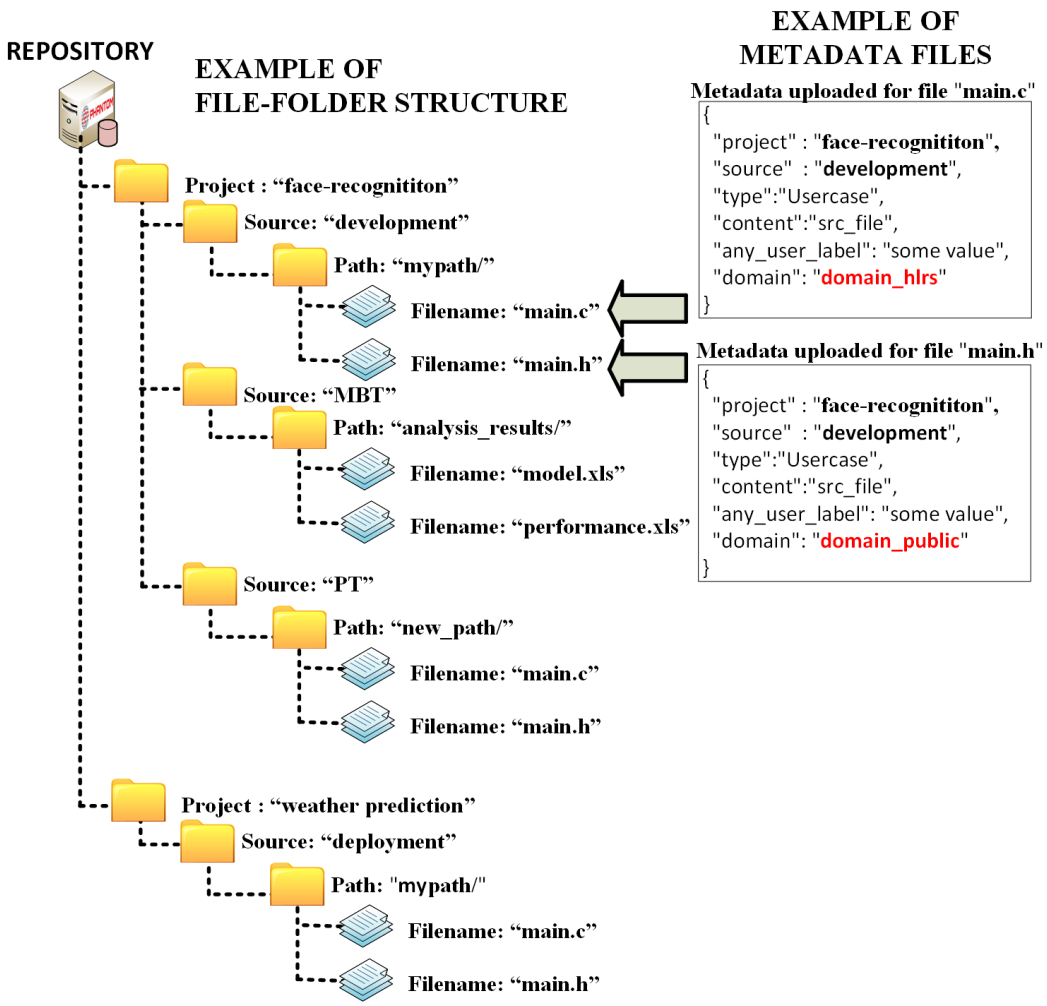


Figure 32. Example of metadata of two uploaded files in the PHANTOM Repository

The Repository Server requests authorization to the PHANTOM Security Framework each time a user wishes to download some file from the Repository. For such requests, the Repository needs the **user_id** which is obtained from the token authentication used by the user, and the **domain** of the file defined in its metadata[8]. Following is an example of requesting authorisation for "**read**" access by the user "**bob@abc.com**" for a file with domain "**domain_hlrs**".

```
curl –XGET http://127.0.0.1:8001/pqapi/access?user=bob@abc.com&ar=read&object=domain_hlrs
```

The Security Server grants the access when the security policy includes the user_id in the domain of file. The response is a string as "**granted**" when effectively the user is included in the file domain, or "**denied**" in other case.

Appendix 4 "Examples of Security Policy Configuration Files" contains two examples of the definition of security policy. The first example shows the definition of two private domains and the default public domain policies, and the definition of domains for some different users.

5.5 SUMMARY OF INNOVATIONS BEYOND SOTA

Innovations include the refinement of the NGAC functional architecture to place the resource access path into the hands of the application developer, lightweight NGAC Policy Server with RESTful Policy Query Interface API and the declarative policy language accepted by the 'ngac' Policy Tool and interpreted by the Policy Server.

[8] The domain is considered as "domain_public" when the domain is not be defined in the metadata.

6. CONCLUSIONS

This deliverable presented the final version of the multi-dimensional optimization technologies, which consists of Multi-Objective Mapper, runtime monitoring functionality and security mechanisms. These technologies combined provide an optimized mapping of application components to heterogeneous platforms.

6.1 SUMMARY OF WORK PERFORMED

According to D5.1 guidelines, the software mechanisms, Multi-Objective Mapper, runtime Monitoring Framework and security mechanisms, described in this deliverable, provide the appropriate APIs and software functions in order to be effectively integrated with PHANTOM use cases. These mechanisms, described in this deliverable, are summarized in the following paragraph:

- Multi-Objective Mapper:
    - **Offline Multi-Objective Mapper**: focusing on eliminating mappings that fail to meet design constraints by testing parts of the system for timing closure
    - **Generic Multi-Objective Mapper:** focusing on the optimal mapping of components and shared data communications throughout the target architecture, towards user defined non-functional requirements, using multi-objective heuristic techniques
- Runtime monitoring
    - Monitoring Server: in charge on metrics storage, and focusses on provide a user interface for monitoring' configuration, and act as interface to access and analyse the run-time collected metrics. Its purpose is support to debug and analyse the behaviour of the user applications.
    - Execution Manager: focus on support on the analysis of previous executions and generation of statistics.
- Security
    - Component network execution integrity: low-level solution that applies to all PHANTOM applications to assure integrity of component network execution on the PHANTOM heterogeneous compute platform
    - Distributed next generation access: high-level solution for unified and flexible access control for systems that span heterogeneous operating environments.

The above mechanisms collaborate to provide a multi-dimensional optimization process considering non-functional requirements, with enhanced accuracy and cognition

provided by the monitoring framework along with enhanced robustness provided by PHANTOM security mechanisms.

REFERENCES

[1] FP7/ICT FiPS project (2013-16): Developing Hardware and Design Methodologies for Heterogeneous Low Power Field, Programmable Servers, available online at: http://cordis.europa.eu/project/rcn/109258_en.html, [Accessed 3 2015].

[2] "FP7/ICT ALMA project (2011-14): Architecture oriented parallelization for high performance embedded Multicore systems using scilAb," [Online]. Available: http://cordis.europa.eu/project/rcn/99828_en.html. [Accessed 3 2015].

[3] "DreamCloud project (2013-16)," [Online]. Available: http://www.dreamcloud-project.org/. [Accessed 8 2018].

[4] "Pareto optimality," [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pareto_efficiency&oldid=855633440. [Accessed 8 2018].

[5] R. I. Davis and A. Burns, "A Survey of Hard Real-time Scheduling for Multiprocessor Systems," vol. 43, no. 4, 2011.

[6] "The Eclipse Modelling Framework," [Online]. Available: http://www.eclipse.org/modeling/emf/. [Accessed 8 2018].

[7] "The Epsilon Project," [Online]. Available: http://www.eclipse.org/epsilon/. [Accessed 8 2018].

[8] "MAST: Modeling and Analysis Suite for Real-Time Applications," [Online]. Available: http://mast.unican.es/. [Accessed 8 2018].

[9] R. Vilalta, . C. V. Apte, . J. L. Hellerstein, S. Ma and S. M. Weiss, Predictive algorithms in the management of computer systems, IBM Systems Journal, 2002.

[10] INCITS 499-2013, Information technology – Next Generation Access Control – Functional Architecture, InterNational Committee for Information Technology Standards, Cyber Security technical committee 1, 2013.

[11] G. Clinton and T. Zachary, ElasticSearch: The Definitive Guide, O'Reilly, 2015.

[12] R. Kuc and M. Rogozinski, ElasticSearch Server, second edition, PACKT publishing, 2014.

[13] B. Dixit, ElasticSearch Essentials Paperback, 2016.

[14] A. Paro, ElasticSearch Cookbook , PACKT publishing, 2013.

[15] G. Radu, M. Lee Hinma and R. Russo, ElasticSearch in Action, ISBN-13: 978-1617291623,ISBN-10: 1617291625, Manning Publications Co, 2016.

APPENDIX 1. EXAMPLES OF EXECUTION OF QUERIES TO THE RESTFUL APIS OF PHANTOM MF SERVER

1. COUNT OF THE EXPERIMENT_IDS of the appid ="demo" and execfile ="pthread-example":

```
curl -s -XGET  http://141.58.0.8:2781/count_executions?appid="demo"\&execfile="pthread-example"
```

```
2
```

2. LIST OF THE EXPERIMENT_IDS of the appid ="demo" and execfile ="pthread-example":

```
curl -s -XGET  http://141.58.0.8:2781/list_executions?appid="demo"\&execfile="pthread-example"
```

```
[
    {
        "key": "AWSSw6NvocAwx GhqUgc",
        "doc_count": 4
    },
    {
        "key": "AWSTAGjEocAwx_GhqUgr",
        "doc count": 4
    }
]
```

3. QUERY OF STATS: (Fields don't appear always in the same order

```
curl -s http://141.58.0.8:2781/get_experiments_stats?appid="demo"\&execfile="pthread-example"\&execution="AWSTAGjEocAwx_GhqUgr";
```

```
{
    "cpu load average": {
        "value": 0
    },
    "disk_read_average": {
        "value": 0
    },
    "average disk write": {
        "value": 8192
    },
    "disk_throughput_average": {
        "value": 819200
    },
    "example of undefined value": {
        "value": null
    },
    "cpu_load_min": {
        "value": 0
    },
    "cpu load count": {
        "value": 1
    },
    "RAM_load_average": {
        "value": 0.07
    },
    "swap_load_average": {
        "value": 0
    },
    "cpu_load_max": {
        "value": 0
    }
}
```

4. QUERY FOR THE DURATION OF THE APPLICATION COMPONENTS

```
curl -s -XGET  http://localhost:8700/get_component_timing?appid="demo"\&execfile="pthread-example"
```


```
{
    "total": 2,
    "max_score": 1,
    "hits": [
        {
            "_index": "demo_pthread-example",
            "_type": "AWSj67hpgiGKJ5c682kq",
            "_id": "AWSj67xzgiGKJ5c682no",
            "_score": 1,
            "_source": {
                "TaskID": "pthread-example",
                "type": "user_defined",
                "host": "node01",
                "local_timestamp": "2018-07-16T18:28:11.730",
                "component_duration": "2007997999",
                "component_name": "component_b",
                "runid": "2018-07-16T18:28:08.222",
                "component_start": "35907401324642",
                "component_end": "35909409322641",
                "server_timestamp": "2018-07-16T18:28:12.785"
            }
        },
        {
            "_index": "demo_pthread-example",
            "_type": "AWSj67hpgiGKJ5c682kq",
            "_id": "AWSj67xzgiGKJ5c682nn",
            "_score": 1,
            "_source": {
                "TaskID": "pthread-example",
                "type": "user_defined",
                "host": "node01",
                "local_timestamp": "2018-07-16T18:28:11.730",
                "component_duration": "2046039708",
                "component_name": "component_a",
                "runid": "2018-07-16T18:28:08.222",
                "component_start": "35907337815227",
                "component_end": "35909383854935",
                "server_timestamp": "2018-07-16T18:28:12.785"
            }
        }
    ]
}
```


5. FULL LIST OF USER METRICS

```
curl -s -XGET  http://localhost:8700/get_user_defined_metrics?appid="demo"\&execfile="pthread-example"
```


```
{
    "total": 10,
    "max_score": 5.1026435,
    "hits": [
        {
            "_index": "demo_pthread-example",
            "_type": "AWSj67hpgiGKJ5c682kq",
            "_id": "AWSj67xzgiGKJ5c682nj",
            "_score": 5.1026435,
            "_source": {
                "TaskID": "pthread-example",
                "type": "user_defined",
                "host": "node01",
                "local_timestamp": "2018-07-16T18:28:10.660",
                "runid": "2018-07-16T18:28:08.222",
                "component_name": "component_a",
                "metric_time": "35908368840116",
                "n_ships_found": "15",
                "server_timestamp": "2018-07-16T18:28:12.785"
            }
```

```
                },
                ....

                ....
                {
                    "_index": "demo_pthread-example",
                    "_type": "AWSj67hpgiGKJ5c682kq",
                    " id": "AWSj67xzgiGKJ5c682nm",
                    " score": 4.0726933,
                    " source": {
                        "TaskID": "pthread-example",
                        "type": "user_defined",
                        "host": "node01",
                        "local timestamp": "2018-07-16T18:28:10.695",
                        "runid": "2018-07-16T18:28:08.222",
                        "component_name": "component_b",
                        "metric_time": "35908404813617",
                        "counter": "1",
                        "server_timestamp": "2018-07-16T18:28:12.785"
                    }
                },
                {
                    "_index": "demo_pthread-example",
                    "_type": "AWSj67hpgiGKJ5c682kq",
                    " id": "AWSj67xzgiGKJ5c682no",
                    " score": 4.0726933,
                    " source": {
                        "TaskID": "pthread-example",
                        "type": "user_defined",
                        "host": "node01",
                        "local timestamp": "2018-07-16T18:28:11.730",
                        "component_duration": "2007997999",
                        "component_name": "component_b",
                        "runid": "2018-07-16T18:28:08.222",
                        "component_start": "35907401324642",
                        "component_end": "35909409322641",
                        "server timestamp": "2018-07-16T18:28:12.785"
                    }
                }
            ]
        }
```


6. QUERY FOR REGISTERED APPLICATIONS (referred as workflows)

```
curl -s -XGET http://localhost:3033/v1/phantom_mf/workflows
```


```
        {   "demo": {
                "application": "demo",
                "author": "new user",
                "optimization": "Time",
                "tasks": [
                    {
                        "device": "demo_desktop",
                        "exec": "hello world",
                        "cores nr": "2"
                    }
                ]
            }
        }
```


7. QUERY FOR A PARTICULAR APPLICATION

```
curl -s -XGET http://localhost:3033/v1/phantom_mf/workflows/demo
```


```
        {       "application":"demo",
                "author":"new user",
                "optimization":"Time",
                "tasks":[{"device":"demo_desktop",
                        "exec":"hello_world",
                        "cores nr":"2"}]
        }
```

8. QUERY FOR LIST PREVIOUS EXECUTIONS (referred as experiments)

```
curl -s -XGET http://localhost:3033/v1/phantom_mf/experiments
```


```
{"AWSoLBjmLgTGqUF8PgdL": {
        "application": "demo",
        "task": "pthread-example",
        "host": "node01",
        "@timestamp": "2018-07-17T14:16:59.617",
        "workflow": "demo"
    }
}, ............
```


9. QUERY FOR THE DESCRIPTION OF A PARTICULAR EXPERIMENT

```
curl -s -XGET http://localhost:3033/v1/phantom_mf/experiments/AWSoLBjmLgTGqUF8PgdL?workflow="demo"
```


```
{       "application":"demo",
        "task":"pthread-example",
        "host":"node01",
        "@timestamp":"2018-07-17T14:16:59.617"
}
```


10. FULL LIST OF USER METRICS

The results shown were obtained when running the example in the Appendix 2.

```
curl -s -XGET http://localhost:3033/v1/phantom_mf/metrics/demo/pthread-example/AWU87m3quRkaX6JeceE9
```


```
[{"simple_function_comp_b":{"count":720,"min":20,"max":60,"avg":39.99950833333334,
    "sum":28799.646000000004},
"number_of_blocks":{"count":720,"min":8,"max":14388,"avg":7198,"sum":5182560},
"counter":{"count":1440,"min":0,"max":719,"avg":359.5,"sum":517680},
"simple_function_comp_a":{"count":720,"min":20,"max":60,"avg":39.99950833333334,
    "sum":28799.646000000004},
"RAM_usage_rate":{"count":969,"min":0.614,"max":0.835,"avg":0.7735675954592356,
    "sum":749.5869999999993},
"swap_usage_rate":{"count":969,"min":0,"max":0,"avg":0,"sum":0},
"runid":{"count":4322,"min":1534332772242,"max":1534332772242,"avg":1534332772242,
    "sum":6631386241629924,"min as string":"2018-08-15T11:32:52.242Z",
    "max as string":"2018-08-15T11:32:52.242Z","avg as string":"2018-08-15T11:32:52.242Z",
    "sum as string":"212110-04-07T13:47:09.924Z"},
"n_ships_found":{"count":720,"min":5,"max":7195,"avg":3600,"sum":2592000},
"CPU_usage_rate":{"count":969,"min":0,"max":500,"avg":27.98658410732714,"sum":27119}}]
```

APPENDIX 2. EXAMPLE OF MONITORING A MULTI-THREAD APPLICATION.

The purpose of this example is to facilitate the integration of the applications with the monitoring environment to those new users without experience. Therefore, this example aims to reduce the time cost of the user for said task.

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include "mf_api.h"
pthread_t tid[2];

#define PI 3.14159265

//3033 and 2779 are the standard and the hlrs-server ports for the monitoring server.
// const char server[]="141.58.0.8:2779";
const char server[]="localhost:3033";
char currentid[100]; //currentid: common UNIQUE id for register in the MF all the threads-metrics in this
execution.

void start_monitoring( const char *server, const char *regplatformid){
/* MONITORING METRICS */
        metrics m_resources;
        m_resources.num_metrics = 3;
        m_resources.local_data_storage = 1; /*remove the file if user unset keep_local_data_flag */
        m_resources.sampling_interval[0] = 10; // 1000 stands for 1s, unit in ms
        strcpy(m_resources.metrics_names[0], "resources_usage");
        m_resources.sampling_interval[1] = 10; // unit in ms
        strcpy(m_resources.metrics_names[1], "disk_io");
        m_resources.sampling_interval[2] = 10; // unit in ms
        strcpy(m_resources.metrics_names[2], "power");
/* MONITORING START */
        mf_start(server, regplatformid, &m_resources);
}

void* doSomeThing( void *args){
        unsigned int i,j, angle=0;
        const int n =720;
        struct Thread_report *my_thread_report;
        my_thread_report = (struct Thread_report *)args;
        my_thread_report->start_time=mycurrenttime();
        my_thread_report->total_metrics= 3;
        my_thread_report->user_label  = (char **) malloc(n * sizeof(char*));
        my_thread_report->user_value  = (char **) malloc(n * sizeof(char*));
        my_thread_report->metric_time = (char **) malloc(n * sizeof(char*));
        for (i=0;i<my_thread_report->total_metrics;i++){
                my_thread_report->user_label[i]  = (char *) malloc(40 * sizeof(char));
                my_thread_report->user_value[i]  = (char *) malloc(40 * sizeof(char));
                my_thread_report->metric_time[i] = (char *) malloc(40 * sizeof(char));
        }

        pthread_t id = pthread_self();
        char component_name[100];
        char temp_i[100];
        char mytime[100];
        for (i=0;i<n;i++){
                llint_to_string_alloc(mycurrenttime(),mytime);// <<-- TIME
                strcat(mytime, ".0");
                angle=(angle+1) % 360;
                for (j=0;j<my_thread_report->total_metrics;j++)
                   strcpy(my_thread_report->metric_time[j], mytime);   // <<-- TIME
                itoa(i,temp_i);
                if(pthread_equal(id,tid[0])) {
                   strcpy(component_name, "first_thread ... ");
                   strcat(component_name, temp_i);
                   printf("\n Processing thread named as: %s\n",component_name);
                   strcpy(my_thread_report->user_label[0], "n_ships_found"); // <<-- LABEL
                   itoa((int) 10*i+5, my_thread_report->user_value[0]);      // <<-- VALUE
                   strcpy(my_thread_report->user_label[2], "simple_function_comp_a"); // <<-- LABEL
                   ftoa((float) (40.0+20.0*cosf(((float) angle)* PI / 180.0)), my_thread_report-
>user_value[2],3);
                } else {
                   strcpy(component_name, "second_thread ... ");
                   strcat(component_name, temp_i);
```

```
                        printf("\n Processing thread named as: %s\n",component_name);
                        strcpy(my_thread_report->user_label[0], "number_of_blocks"); // <<-- LABEL
                        itoa((int) 20*i+8, my_thread_report->user_value[0]);         // <<-- VALUE
                        strcpy(my_thread_report->user_label[2], "simple_function_comp_b"); // <<-- LABEL
                        ftoa((float) (40.0+20.0*sinf(((float) angle)* PI / 180.0)), my_thread_report-
>user_value[2],3);
                    }
                    strcpy(my_thread_report->user_label[1], "counter"); // <<-- LABEL
                    itoa((int) i, my_thread_report->user_value[1]);     // <<-- VALUE
                    user_metrics_buffer(currentid,*my_thread_report);
                    usleep(10000);  /* sleep unit is on us */
            }
            printf("\n Finishing thread named as: %s\n",component_name);
            fflush(stdout);
            my_thread_report->end_time=mycurrenttime();
    return NULL;
}

int main(int argc, char* argv[]) {
            const int amount_of_threads = 2; //we have 2 threads in this example
            /********************MONITORNG START ************************/
            struct Thread_report all_thread_reports[2];
            char regplatformid[]="node01";
            char appid[]="demo";
            char execfile[]="pthread-example";
            start_monitoring(server, regplatformid);
            strcpy(all_thread_reports[0].taskid,"component_a");
            strcpy(all_thread_reports[1].taskid,"component_b");
            for(int i=0;i<amount_of_threads;i++)
                    all_thread_reports[i].total_metrics=0;
            if(argc>1)
                    strcpy(currentid,argv[1]);
            else
                    strcpy(currentid,"missingid");
            /*************************************************************/
            for(int i=0;i<amount_of_threads;i++){
            int err = pthread_create(&(tid[i]), NULL, &doSomeThing, (void *) (&all_thread_reports[i]));
            if (err != 0)
               printf("\n Can't create thread :[%s]", strerror(err));
            else
               printf("\n Thread created successfully\n");
            }
            for(int i=0;i<amount_of_threads;i++)
              (void) pthread_join(tid[i], NULL);
            printf("\n Finishing program\n");
            /************* MONITORING END ******************************/
            register_workflow( server, regplatformid, appid, execfile);
            for(int i=0;i<amount_of_threads;i++)
                    register_end_component(currentid, all_thread_reports[i]);
            monitoring_send( server, appid, execfile, regplatformid);
            mf_end();
            for(int i=0;i<amount_of_threads;i++){
               printf(" Execution label of the workflow: \"%s\"\n", currentid);
               printf("   THREAD num %i, name : %s\n",i, all_thread_reports[i].taskid);
               printf("     total metrics %i:\n",all_thread_reports[i].total_metrics);
               for(int j=0;j<all_thread_reports[i].total_metrics;j++)
                   printf("   %s : %s\n", all_thread_reports[i].user_label[0],
all_thread_reports[i].user_value[0]);
               printf("   Start time: %llu ns\n", all_thread_reports[i].start_time);
               printf("   End time  : %llu ns\n\n", all_thread_reports[i].end_time);
            }
            /*************************************************************/
            for (int i=0;i<amount_of_threads;i++){
               for (int j=0;j<all_thread_reports[i].total_metrics;j++ ){
                      free(all_thread_reports[i].user_label[j]);
                      free(all_thread_reports[i].user_value[j]);
                      free(all_thread_reports[i].metric_time[j]);
               }
               free(all_thread_reports[i].user_label);
               free(all_thread_reports[i].user_value);
               free(all_thread_reports[i].metric_time);
            }
            return 0;
}
```

The expected output on the screen should look like:

```
Execution label of the workflow: "2018-08-15T11:32:52.242"
   THREAD num 0, name : component_a
     total metrics 3:
   n_ships_found : 7195
   n_ships_found : 7195
   n_ships_found : 7195
   Start time: 5251874110832 ns
   End time  : 5261328626883 ns

 Execution label of the workflow: "2018-08-15T11:32:52.242"
   THREAD num 1, name : component_b
     total metrics 3:
   number_of_blocks : 14388
   number_of_blocks : 14388
   number_of_blocks : 14388
   Start time: 5251929702080 ns
   End time  : 5261395373615 ns
```

The example register thousands of metrics at the MF-Server, which in JSON format should look like:


```
…
   }, {
      "_index" : "demo_pthread-example",
      "_type" : "AWU87m3quRkaX6JeceE9",
      "_id" : "AWU87oPQuRkaX6Jecea-",
      "_score" : 1.0,
      "_source" : {
        "TaskID" : "pthread-example",
        "type" : "user_defined",
        "host" : "node01",
        "local_timestamp" : "2018-08-15T11:32:54.653",
        "runid" : "2018-08-15T11:32:52.242",
        "component_name" : "component_b",
        "metric_time" : "5252788258303.0",
        "number_of_blocks" : 1468,
        "server_timestamp" : "2018-08-15T11:33:08.943"
      }
    }, {
      "_index" : "demo_pthread-example",
      "_type" : "AWU87m3quRkaX6JeceE9",
      "_id" : "AWU87oPQuRkaX6Jecea_",
      "_score" : 1.0,
      "_source" : {
        "TaskID" : "pthread-example",
        "type" : "user_defined",
        "host" : "node01",
        "local_timestamp" : "2018-08-15T11:32:54.655",
        "runid" : "2018-08-15T11:32:52.242",
        "component_name" : "component_a",
        "metric_time" : "5252790121507.0",
        "n_ships_found" : 735,
        "server_timestamp" : "2018-08-15T11:33:08.943"
      }
    }, {
      "_index" : "demo_pthread-example",
      "_type" : "AWU87m3quRkaX6JeceE9",
      "_id" : "AWU87oPhuRkaX6JecebD",
      "_score" : 1.0,
      "_source" : {
        "TaskID" : "pthread-example",
        "type" : "user_defined",
        "host" : "node01",
        "local_timestamp" : "2018-08-15T11:32:54.658",
        "runid" : "2018-08-15T11:32:52.242",
        "component_name" : "component_a",
        "metric_time" : "5252790121507.0",
        "simple_function_comp_a" : 45.512,
        "server_timestamp" : "2018-08-15T11:33:08.960"
      }
….
```


Source code available at:

https://github.com/PHANTOM-Platform/Monitoring/tree/master/example-monitoring-app-with-pthreads

APPENDIX 3. EXAMPLES OF SECURITY POLICIES

A graphical representation of two attribute-based policies are shown in Figure 33. Policy (a) defines the Project Access policy class that aggregates users by groups and objects by projects. Access permissions to project objects are expressed by groups relative to projects. Policy (b) defines a File Management policy class that illustrates how access to personal files may be restricted or shared.

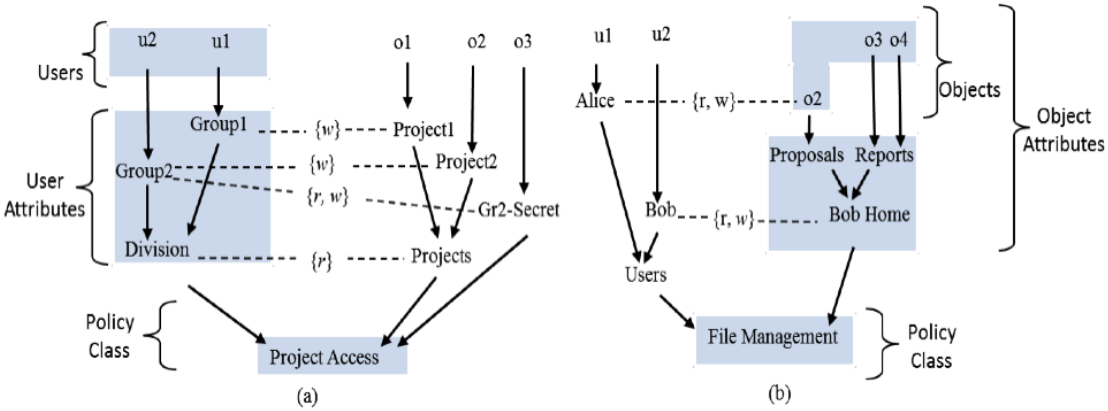


Figure 33: Two attribute-based security policies

Assignment relations are represented as arrows. Association relations are represented by a dashed line and include the set of access rights shown on the line. The derived privileges (allowed user-operation-object tuples) of each of the policies separately are shown in

(u1,r,o1),(u1,w,o1),(u1,r,o2),(u2,r,o1), (u2,r,o2),(u2,w,o2),(u2,r,o3),(u2,w,o3)	(u1,r,o2),(u1,w,o2),(u2,r,o2),(u2,w,o2), (u2,r,o3),(u2,w,o3),(u2,r,o4),(w2,w,o4)

Figure 34.

(u1,r,o1),(u1,w,o1),(u1,r,o2),(u2,r,o1), (u2,r,o2),(u2,w,o2),(u2,r,o3),(u2,w,o3)	(u1,r,o2),(u1,w,o2),(u2,r,o2),(u2,w,o2), (u2,r,o3),(u2,w,o3),(u2,r,o4),(w2,w,o4)

Figure 34: Derived privileges of policies (a) and (b)

A run of the ngac policy tool is shown in Figure 35. The first column shows the declarative policy language specification of Policy (a). The next three columns show test of some internal functions, including the logical object space visible to the different users. The last column shows the access control tests, which are answered by `permit` or `deny`. The `permit` response by ngac to the first query, `access('Policy (a)',(u1,r,o1))`, is highlighted. This response corresponds to the path highlighted in the graphical representation of the policy, specifically, user `u1` belongs to user attribute `Group1` which in turn belongs to `Division`, which is granted `r` access to object attribute `Projects`, which contains `Project1`, which in turn contains object `o1`.

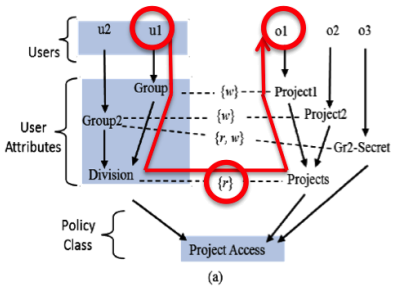
The PM server receives direction from the PM admin tool in the form of imperative commands that build or modify the security policy database. The ngac policy tool can also be able to convert its declarative policy description to the imperative command form used by the PM server reference implementation. This provides the ability to develop a policy using the ngac policy tool and to deploy it in the PM server.

```
ngac> proc(testA,verbose).
> newpol('Policy (a)').
> los('Policy (a)').
user(u1)
user(u2)
object(o1)
object(o2)
object(o3)
user_attribute(Division)
user_attribute(Group1)
user_attribute(Group2)
object_attribute(Gr2-Secret)
object_attribute(Project1)
object_attribute(Project2)
object_attribute(Projects)
policy_class(Project Access)
assign(u1,Group1)
assign(u2,Group2)
assign(Group1,Division)
assign(Group2,Division)
assign(o1,Project1)
assign(o2,Project2)
assign(o3,Gr2-Secret)
assign(Project1,Projects)
assign(Project2,Projects)
assign(Division,Project Access)
assign(Projects,Project Access)
assign(Gr2-Secret,Project Access)
associate(Group1,[w],Project1)
associate(Group2,[w],Project2)
associate(Group2,[r,w],Gr2-Secret)
associate(Division,[r],Projects)
Connector(PM)
```

```
> dps('Policy (a)').
Group1,r,Project1
Group1,r,Project2
Group1,r,o1
Group1,r,o2
Group2,r,Project1
Group2,r,Project2
Group2,r,o1
Group2,r,o2
u1,r,Project1
u1,r,Project2
u1,r,o1
u1,r,o2
u1,w,o1
u2,r,Project1
u2,r,Project2
u2,r,o1
u2,r,o2
u2,r,o3
u2,w,o2
u2,w,o3

> aoa(u1).
Project1
Project2
Projects
o1
o2

> minaoa(u1).
Projects
```

```
> userlos('Policy (a)',u1).
V= PM
 Project Access
 o1
 o2
 Project1
 Project2
 Projects
E= Project Access,PM
 Projects,Project Access
 o1,Project1
 o2,Project2
 Project1,Projects
 Project2,Projects

> aoa(u2).
Gr2-Secret
Project1
Project2
Projects
o1
o2
o3

> minaoa(u2).
Gr2-Secret
Projects
```

```
> userlos('Policy (a)',u2).
V= PM
 Project Access
 o1
 o2
 o3
 Project1
 Project2
 Gr2-Secret
 Projects
E= Project Access,PM
 Gr2-Secret,Project Access
 Projects,Project Access
 o1,Project1
 o2,Project2
 o3,Gr2-Secret
 Project1,Projects
 Project2,Projects
```




```
ngac> proc(queryA,verbose).
> access('Policy (a)', (u1,r,o1)).
permit
> access('Policy (a)', (u1,w,o1)).
permit
> access('Policy (a)', (u1,r,o2)).
permit
> access('Policy (a)', (u1,w,o2)).
deny
> access('Policy (a)', (u1,r,o3)).
deny
> access('Policy (a)', (u1,w,o3)).
deny
> access('Policy (a)', (u2,r,o1)).
permit
> access('Policy (a)', (u2,w,o1)).
deny
> access('Policy (a)', (u2,r,o2)).
permit
> access('Policy (a)', (u2,w,o2)).
permit
> access('Policy (a)', (u2,r,o3)).
permit
> access('Policy (a)', (u2,w,o3)).
permit
ngac>
```

Figure 35: Description of Policy (a) and interactive test

APPENDIX 4: EXAMPLES OF SECURITY POLICY CONFIGURATION FILES

Below appear an example of the definition of two private domains and the public domain policies (file named as demo_repository_policy.pl). In the example are defined 4 users, two belonging to each private domain.

After that, appear the files **simons_ngac_policy.pl** and **simons_ngac_script.pl** used for defining the policy used with the GMV use case.

File: demo_repository_policy.pl

```
policy('Policy (phantom_demonstration)','File Management', [
    /** Define the users */
             user('user1@hlrs.de'),
             user('user2@hlrs.de'),
             user('user3@opengroup.org'),
             user('user4@opengroup.org'),
    /** Define the type of objects in the repository*/
             object('domain_public'),
             object('domain_hlrs'),
             object('domain_opengroup' ),
    /** Assign to the users to global set */
             user_attribute('Users'),
             user_attribute('Users_hlrs'),
             user_attribute('Users_opengroup'),
    /** Assign to the users to groups */
             assign(user1@hlrs.de','Users_hlrs'),
             assign('user2@hlrs.de','Users_hlrs'),
             assign(user3@opengroup.org','Users_opengroup'),
             assign('user4@opengroup.org','Users_opengroup'),
    /** groups */
             assign('Users_hlrs','Users'),
             assign('Users_opengroup','Users'),
    /** Assign all the variables to the class 'File Management', needed by the tool*/
             policy_class('File Management'),
             connector('PM'),
             assign('Users','File Management'),
             assign('domain_public','File Management'),
             assign('domain_hlrs','File Management'),
             assign('domain_opengroup','File Management'),
    /** define-add the permissions of each group on each type of object */
             operation(r,'File'),
             operation(w,'File'),
             associate('Users',[r,w],'domain_public'),
             associate('Users_hlrs',[r,w],'domain_hlrs'),
             associate('Users_opengroup',[r,w],'domain_opengroup')
       ]).
```

The next files **simons_ngac_policy.pl** and **simons_ngac_script.pl** define relation between users' domains and registered contents in the Repository with the GMV use case. The Figure 36 aims to represent that relation.

File: **simons_ngac_policy.pl**

```
policy('review', 'Surveillance Database', [

      % list of users
      user('satellite'), % producer of the image and related info
      user('geo_entity'), % producer of earth geographic info
      user('simons_app'), % program that will access the available data
      user('intruder'), % menace
      user_attribute('Producer'), % able to read and write any file
      user_attribute('Geo_Producer'), % able to read and write any geographic
file
      user_attribute('Consumer'), % able to read any file
      user_attribute('Users'),

      % list of objects
      object('metadata'),
      object('image_data'),
      object('coastline'),
      object_attribute('Image'),
      object_attribute('Aux file'),
      object_attribute('Data'),

      policy_class('Surveillance Database'),
      connector('pm'),

      % list of relations of users
      assign('satellite', 'Producer'),
      assign('geo_entity', 'Geo_Producer'),
      assign('simons_app', 'Consumer'),
      assign('satellite', 'Users'),
      assign('simons_app', 'Users'),
      assign('geo_entity', 'Users'),

      % list of relations of objects
      assign('metadata', 'Image'),
      assign('image_data', 'Image'),
      assign('coastline', 'Aux file'),
      assign('Image', 'Data'),
      assign('Aux file', 'Data'),

      % other relations
      assign('Users', 'Surveillance Database'),
      assign('Data', 'Surveillance Database'),

      % access policies
      associate('Producer', [read, write], 'Data'),
      associate('Geo_Producer', [read, write], 'Aux file'),
      associate('Consumer', [read], 'Data')
]).
```

File: **simons_ngac_script.pl**

```
import(policy('simons_ngac_policy.pl')).
newpol('review').

access('review', (satellite, read, metadata)). % permit
access('review', (satellite, write, metadata)). % permit
access('review', (satellite, read, image_data)). % permit
access('review', (satellite, write, image_data)). % permit
access('review', (satellite, read, coastline)). % permit
access('review', (satellite, write, coastline)). % permit

access('review', (simons_app, read, metadata)). % permit
access('review', (simons_app, write, metadata)). % deny
access('review', (simons_app, read, metadata)). % permit
access('review', (simons_app, write, metadata)). % deny
access('review', (simons_app, read, coastline)). % permit
access('review', (simons_app, write, coastline)). % deny

access('review', (geo_entity, read, metadata)). % deny
access('review', (geo_entity, write, metadata)). % deny
access('review', (geo_entity, read, image_data)). % deny
access('review', (geo_entity, write, image_data)). % deny
access('review', (geo_entity, read, coastline)). % permit
access('review', (geo_entity, write, coastline)). % permit

access('review', (intruder, read, metadata)). % deny
access('review', (intruder, write, metadata)). % deny
access('review', (intruder, read, image_data)). % deny
access('review', (intruder, write, image_data)). % deny
access('review', (intruder, read, coastline)). % deny
access('review', (intruder, write, coastline)). % deny
```

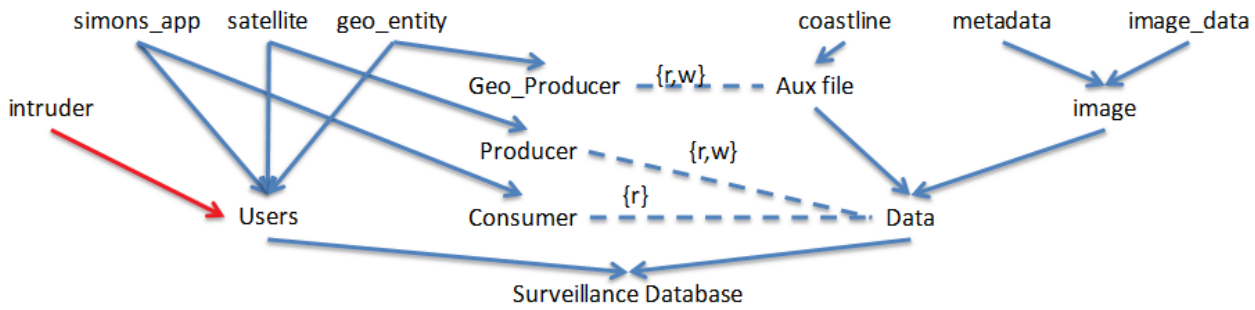


Figure 36. Schema of the policy used with the GMV use case.