PHANTOM

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

Project Number 688146

D2.1 – First report on system software for multi-dimensional optimization on heterogeneous systems

Version 2.0
3 November 2017
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 Phone: +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	22/03/2017
0.2	Integration and review of WINGS, YORK, USTUTT-HLRS contributions	13/04/2017
0.3	Updates and second review of YORK, USTUTT-HLRS contributions, integration and review of TOG contribution	05/05/2017
0.4	Integration and review of updates of USTUTT-HLRS and TOG after second review	22/05/2017
1.0	Updates after internal review of GMV and EGM and final version preparation	07/06/2017
2.0	Updates to address recommendations from M18 EC review	03/11/2017

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The deliverable D2.1-"*First report on system software for multi-dimensional optimization on heterogeneous systems*" describes the first prototypes of the multi-objective mapper, runtime monitoring functionality and security mechanisms which constitute the basic mechanisms to be considered for the integration and further refinements towards the multi-dimensional optimization process.

The Multi-Objective Mapper (MOM) is responsible for the optimal mapping of components and shared data communications throughout the target architecture, towards user defined non-functional requirements, while guaranteeing Quality of Service. Two complementary approaches and mechanisms for MOM are proposed.

First, the Offline MOM relies on analytical approaches to verify the timing constraints of (parts of) a deployment without runtime testing, eliminating mappings that fail to meet design constraints by testing the worst case response times of parts of the system, especially for applications with real-time requirements such as the Telecoms and Surveillance use cases. The implementation comprises a metamodelling phase for describing the inputs of the Offline MOM including the platform model, a model pattern matching phase to extract parts of the model that can be analyzed with real-time analysis, and use of open-source suite of tools for performing timing and schedulability analysis.

Second, the Generic MOM considers evolutionary multi-objective optimization approaches, inspired by genetic algorithms, in order to optimize the placement of components against multiple objectives such as power, performance, security, taking into account requirement/policies from application developers' side, as well as the status and capabilities of computing resources. An optimized two-level mutation loop is executed to populate the Pareto optimal solutions. Currently, Generic MOM tool estimates the total computation time, the total communication time and the total execution time, based on the metrics related to CPU speed, memory size and memory access time.

The runtime monitoring functionality plays an essential role for the application optimization based on the understanding of both software non-functional properties and hardware quality attributes with regards to performance and energy consumption aspects. The PHANTOM monitoring framework architecture follows the design of ATOM – the monitoring solution elaborated by the EU projects EXCESS and DreamCloud – comprising a client-server architecture, collecting runtime information of the monitored hardware resources by means of a monitoring agent service and transmitting to a centralised monitoring server. However, ATOM has been considerably redesigned for the PHANTOM sake, aiming along with enabling the application-level monitoring and the more modular component-based and service-oriented architecture.

In terms of security mechanisms, PHANTOM addresses two general security problems linked to the project's specific innovation, namely the seamless integration and orchestration of heterogeneous computing elements while exercising control over the qualities exhibited by their combined operation over a broad range of scale.

First, Isolation of resources and Information Flow Control among them (IIFC) is provided in both the PHANTOM development platform and the deployment platform.

IIFC is provided in the PHANTOM development platform by the operating environment in which the tools of the PHANTOM system are run. IIFC is provided in the PHANTOM deployment environment by a combination of mechanisms in the operating environment, by the deployment manager, and by MILS-style software/firmware/hardware foundational components in terms of resource managers, namely the runtime managers of the physical computational resources (processors and memories) that permit safe sharing of those resources.

Second, coherent, uniform and flexible access control over resources for large-scale PHANTOM applications within a distributed and heterogeneous operating environment, is achieved by a decomposition of the reference monitor concept, entitled as “Policy Machine” (PM), inspired by the NGAC standard and extending the NGAC reference implementation. Necessary extensions to the PM reference implementation comprise support for heterogeneous processing elements, heterogeneous operating environments, micro-services and Web services for enterprise class PHANTOM applications, as well as interfaces for PHANTOM tool components. Extensions also possibly include support and interfaces for lower level access control policy support for embedded systems, covering both static and dynamic deployment possibilities.

The description for each of the above technologies consists of the relevant use case requirements (with potential link with D1.1). Then, it includes the design specifications for each technology developed in WP2 in relation to the PHANTOM tool-flow and the PHANTOM Programming model, as specified in D1.2. In addition, this deliverable provides extended implementation details including the programming tools that are used from each technology with potential link to D4.1, which provides a more detailed insight of the programming tools. After the implementation details, the description of each technology continues with the available testing results if exist (relevant KPIs can be found in the deliverable D5.1), the dependencies with other PHANTOM mechanisms as introduced in D1.2 and the next steps regarding the implementation and the integration process, towards the deliverable D2.2-" *Final report on system software for multi-dimensional optimization on heterogeneous systems* ".

Finally, the conclusions section presents the summary of the presented technologies, the achieved level of this preliminary integration procedure, and the future steps of the implementation and integration activities. The extensive description of the final versions of the addressed technologies will be presented in the deliverable D2.2-" *Final report on system software for multi-dimensional optimization on heterogeneous systems*" at M30.

1. INTRODUCTION

1.1 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 the energy consumption of multi-core processing platforms. A great challenge is the exploration of the available hardware infrastructure beyond the homogeneous multi-core platforms, to heterogeneous platforms including CPU/GPU, CPU/FPGA and embedded/cloud platforms. PHANTOM multidimensional 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 multidimensional 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 FiPS [1] project focuses on reducing the power performance ratio within data-centers by improving the usability and usefulness of heterogeneous platforms considering FPGAs, embedded CPUs, GPUs and multicore accelerators in high-performance and low-power heterogeneous computing servers. Furthermore, the ALMA project [3] 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 in DreamCloud project [2], which provides a 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, Multi-Objective Optimization technologies in the context of PHANTOM consider additional non-functional properties such as memory utilization, data security, temperature, as well as design principles like modularity, predictability, robustness and data-movement. Furthermore, the optimization mechanisms will consider resources and requirements from heterogeneous platforms (e.g. GPU, FPGA) in a unified manner, able to split application and map 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. In addition, the optimization process will be assisted by Model Based Testing mechanisms, which will evaluate Quality of Service at the functional level regarding the application execution and also provide 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 among 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 an advanced monitoring framework able to monitor several types of resources (e.g. computation, I/O, storage,

network and interconnect infrastructures) in a unified manner, offering on-the-fly data analysis, prediction and visualization capabilities that will further interact with the mapping mechanisms 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 which 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 authorization 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

This document is the first deliverable of the Work package 2, with main objectives: 1) the design and implementation of a multi-objective self-adaptive mapper for optimally mapping application components to heterogeneous hardware platforms over the computing continuum, 2) the implementation of the runtime monitoring process for several types of resources in a unified manner offering configuration, data analytics and visualization/supervision capabilities and 3) the implementation of security mechanisms across all levels starting from isolated protection domains supported by hardware and software. The purpose of this document is to provide a first description of the software mechanisms which constitute the multi-dimensional optimization process targeting systems with increased heterogeneity. This first description capitalizes on information and details that are further described in existing PHANTOM deliverables, as described in the following paragraph:

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

In addition, developed technologies/components in WP2 need to interact under the specified in D1.2 toolflow 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 relevant 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 available infrastructure resources to pick 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 able to perform runtime monitoring of several types of both application and system resources storing 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, visualization by means of data correlation and prediction of upcoming states and events (this will be addressed in D2.2).

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

The aforementioned technologies include appropriate methods and APIs targeting their integration to a holistic multi-dimensional optimization framework and the overall PHANTOM platform at the end, following the WP5-"*Integration, use case implementation and validation*" integration directives and finally their refined versions and updates will be described in deliverable D2.2 " *Final report on system software for multi-dimensional optimization on heterogeneous systems* ".

2. MULTI-OBJECTIVE MAPPER

2.1 OFFLINE MOM

As detailed in deliverable D1.2, the purpose of the Multi-Objective Mapper is to constrain all previously unconstrained aspects of a deployment. Recall that the 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 MOM is to assign all unconstrained mappings to create a complete deployment.

The Generic MOM described in section 2.2 does this through the use of an evolutionary 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 is different, in that it relies on analytical approaches to determine ahead-of-time the timing properties of a given deployment.

The Offline MOM is an optional part of the PHANTOM toolflow, focused on eliminating mappings that fail to meet design constraints by testing the worst case response times of parts of the system. It can be used to verify the timing constraints of (parts of) a deployment without runtime testing.

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 which 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.

2.1.1 Use case 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, U51, U54, U55, U57, U60, U61 and U63.

Req. No.	Requirement	Overall Priority
U49	The mapping proposed by the PHANTOM framework shall reason about the functional and non-functional requirements of the application	SHALL
U50	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for execution time	SHOULD
U51	The PHANTOM framework should provide facilities	SHOULD

Req. No.	Requirement	Overall Priority
	(e.g. APIs or annotations) to consider non-functional requirements for memory	
U54	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for communications bandwidth	SHOULD
U55	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for I/O	SHOULD
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
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
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
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

2.1.2 Design specifications

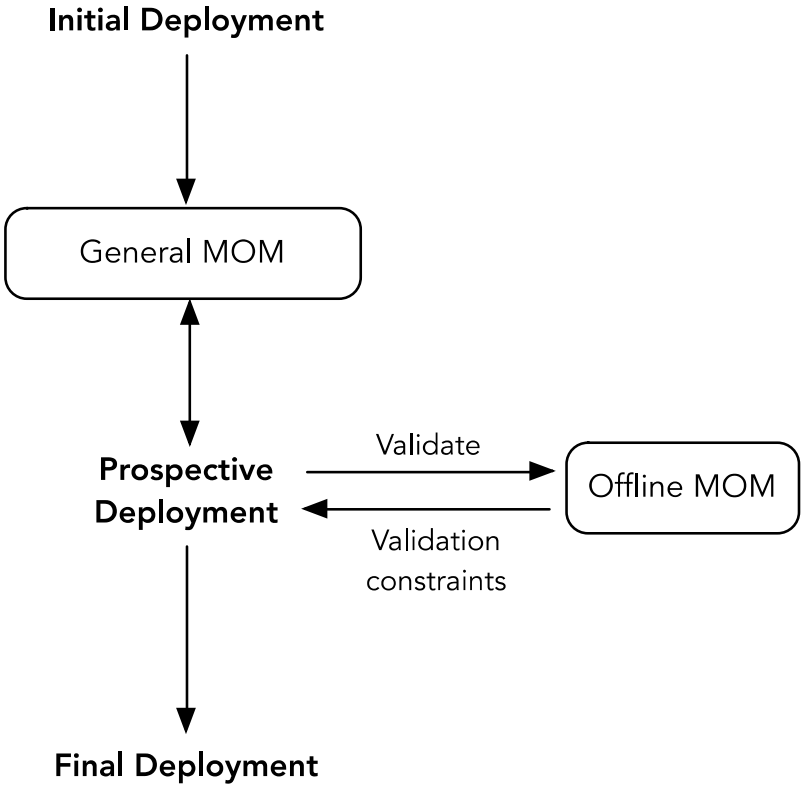


Figure 1: **The Offline and General MOM toolflow**

Figure 1 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 General MOM is then used to refine this into a fully-specified prospective deployment, in which all components are speculatively mapped. The developer can then optionally 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 prevent the current deployment and the General 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 [4]. It is the work of the Offline MOM to identify the scenarios when existing analysis can be applied.

2.1.3 Implementation details

2.1.3.1 Metamodelling

The Offline MOM is powered by a metamodelling approach based on the Eclipse Modelling Framework [5] and Epsilon [6]. 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 2.

Figure 2: **A fragment of the Ecore metamodel and an instantiation in XML format**

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

2.1.3.2 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 analyzed 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 which 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 sporadics 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.

2.1.3.3 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 [7]. MAST (Modeling and Analysis Suite for Real-Time Applications) is an open-source suite of tools for performing various kinds of timing and schedulability analysis. 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 transformations. 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.

2.1.4 Analysis Results and Feedback

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 to try another as shown in the toolflow in Figure 1.

2.1.5 Dependencies/integration aspects

The Offline MOM is a part of the Generic MOM toolflow as described in figure 1 of deliverable D2.1. It will be integrated into the same part of the toolflow.

2.1.6 Innovations beyond the state-of-the-art

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 Generic MOM (Section 2.2) 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 which attempts to consider the entire system as a whole. This still relies on a communications platform/protocol which has determinable worst-case performance guarantees.

2.1.6.1 Background technologies utilised in development

The Offline MOM uses the Epsilon suite of tools to perform model matching, model transformation, and model to text transformations.

The work uses the MAST framework to perform the actual real-time analysis.

2.1.6.2 Summary of new technologies/extensions developed

The formats for the Component Network, Platform Description, and Deployment Plan were defined and refined for the requirements of the platform.

Integration scripts and programs are implemented to place the Offline MOM into the development flow. The scripts read a MOM deployment, perform the appropriate model transformations, invoke the analysis framework, and parse the results.

An initial set of model patterns and model transformations are implemented.

2.1.6.3 Early/Full Prototypes functionality

Early-First Year Prototype:

Initial model patterns and transformations are defined and implemented to support fixed priority pre-emptive scheduling, deadline-based scheduling, and simple bandwidth analysis.

Full Prototype and Next Steps:

It is also intended to explore more complex architectures and on-chip networks by modelling the shared objects from the Component Network as sporadic servers and analysing each CPU separately using an interval algebra-based approach [8] developed in the EU DreamCloud project [9].

The Telecoms and Surveillance use cases have strict timing requirements and so can be used to demonstrate the benefits of integrated timing analysis. To augment this for academic dissemination, it is also intended to perform analysis based on an autonomous vehicle platform illustrated in Figure 3.

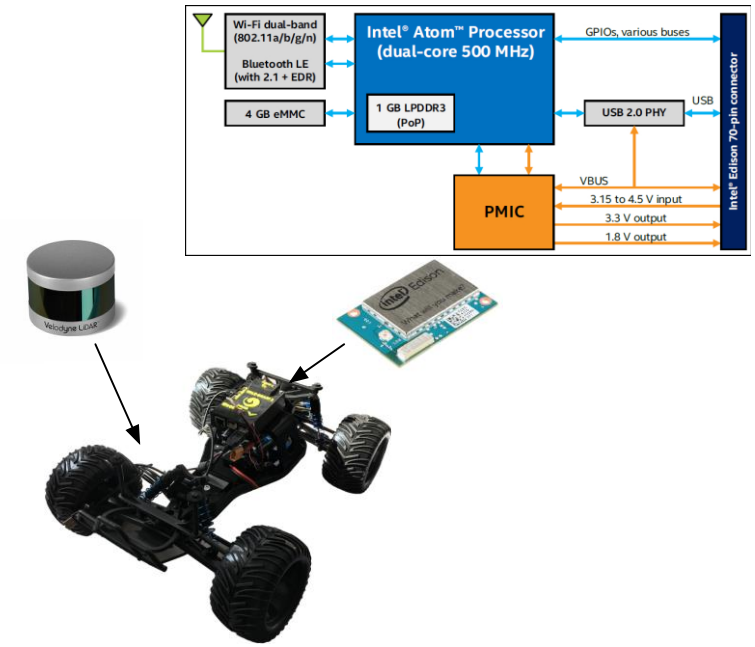


Figure 3: **Additional Dissemination Platform**

The platform is based on the Intel Edison SoC, a platform for developing Internet of Things applications. The Edison is a three-core design; a dual-core standard CPU alongside a single core microcontroller. The architecture is very similar to existing use cases so any porting effort will be small. Software on the Edison uses the LIDAR to determine the features of the surroundings and take appropriate action. There is a range of software tasks to be performed, some soft-real time, and some hard-real time.

The value of this platform is that it is power-constrained because it is powered by batteries, thereby allowing the evaluation of the power optimisation aspects of the PHANTOM toolchain.

2.2 GENERIC MOM

The Multi-Objective Mapper (MOM) is responsible for the optimal mapping of components and shared data communications throughout the target architecture, towards user defined non-functional requirements, while guaranteeing Quality of Service. MOM considers evolutionary 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) and temperature, taking into account requirement/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.

2.2.1 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, some of the requirements referring to all use cases are the following:

- 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.

Req. No.	Requirement	Overall Priority
U48	The PHANTOM framework shall propose a mapping of the parallel code blocks, composing the application, to the available resources/target platforms	SHALL
U49	The mapping proposed by the PHANTOM framework shall reason about the functional and non-functional requirements of the application	SHALL
U50	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for execution time	SHOULD
U51	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for memory	SHOULD
U52	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for power consumption	SHOULD
U53	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for cost	SHOULD
U54	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for communications bandwidth	SHOULD
U55	The PHANTOM framework should provide facilities (e.g. APIs or annotations) to consider non-functional requirements for I/O	SHOULD
U56	The PHANTOM framework should provide facilities	SHOULD

Req. No.	Requirement	Overall Priority
	(e.g. APIs or annotations) to consider non-functional requirements for security	
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
U59	The user shall be able to enforce a given mapping, overriding the proposed one depending on existing constraints and availabilities	SHALL
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
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
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

2.2.2 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, data security and other use case requirements, as defined in D1.1. To that respect, MOM needs to interact with other components in PHANTOM architecture, 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.

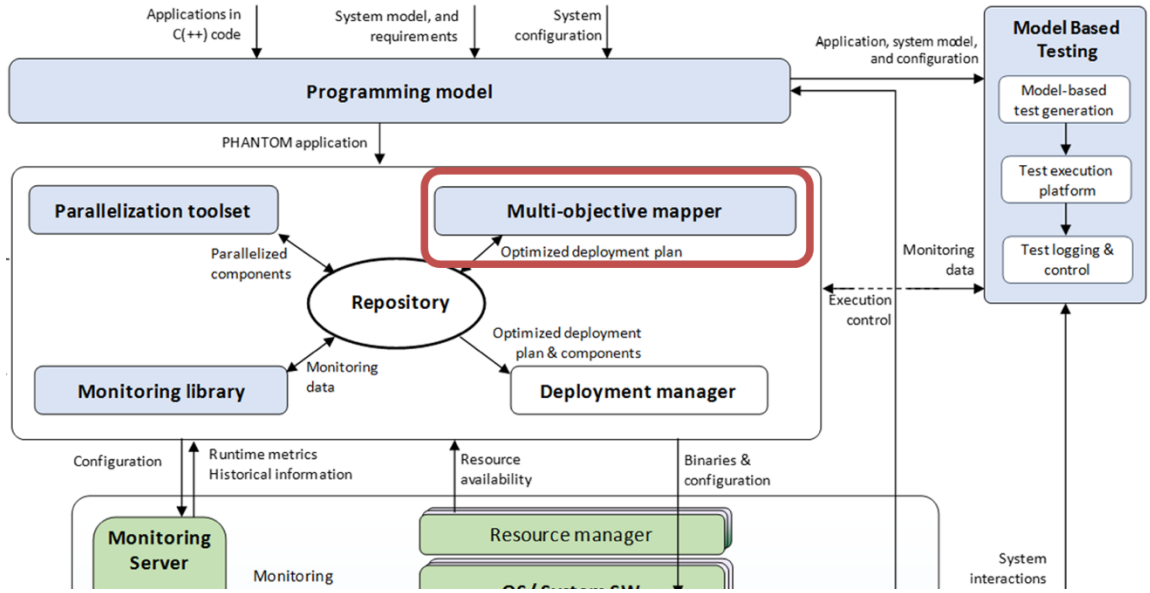


Figure 4: **MOM positioning in PHANTOM architecture and toolflow**

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 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).

2.2.2.1 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 multitude of application components, communication objects, and relations between communication objects and components defining the source and destination components.

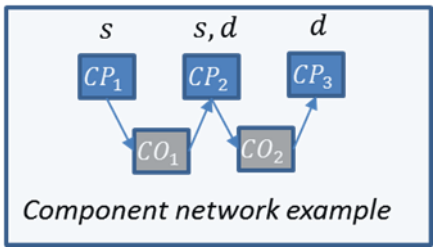


Figure 5: Component network example

Assuming a set of application components:

$$\boldsymbol{CP} = < \boldsymbol{CP_1}, \ \ldots, \boldsymbol{CP_N} > \quad (1)$$

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

$$\boldsymbol{CO} = < \boldsymbol{CO_1}, \ \ldots, \boldsymbol{CO_M} > \quad (2)$$

where M is the number of communication objects.

We define a relation:

$$\boldsymbol{P}: \boldsymbol{CO} \rightarrow \boldsymbol{CP} \quad (3)$$

s.t. we have a subset

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

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):

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

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

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

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

$$\boldsymbol{R}: \boldsymbol{PE} \rightarrow \boldsymbol{MEM}, \quad (7)$$

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

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

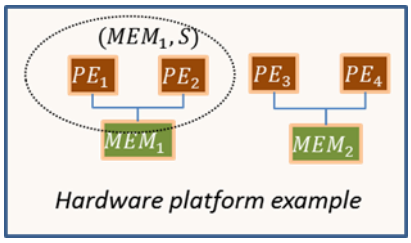


Figure 6: Example of hardware platform

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:

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

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:

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

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:

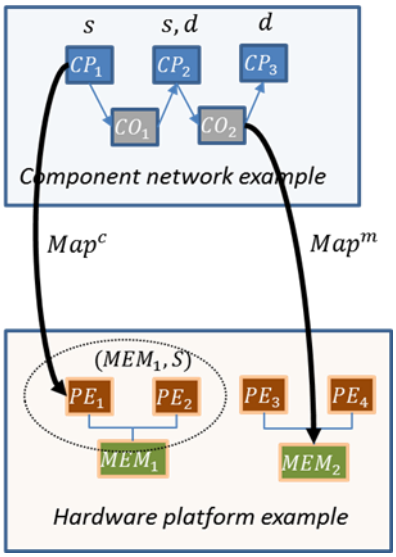


Figure 7: Example of mapping

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 component network, where, based on a mapping $m \in Map$, and considering cp_1 and cp_2, with $cp_1 \overset{m}{\rightarrow} pe$ and $\forall\, co \in CO' : (co, cp_1, cp_2) \in P$,

if

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

then

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

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

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

Let $mem \in MEM$,

if

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

Then

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

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

Let

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

the vector of maximum memory sizes and

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

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

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

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

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

and the Total energy consumption, given a mapping m:

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

Same constraints can be defined per component

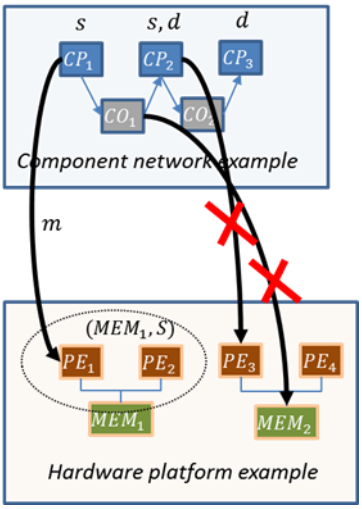


Figure 8: Example of constraint mapping operation

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:

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

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 or 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

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

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

2.2.3 Implementation details

The optimized mapping is generated from an evolutionary 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:

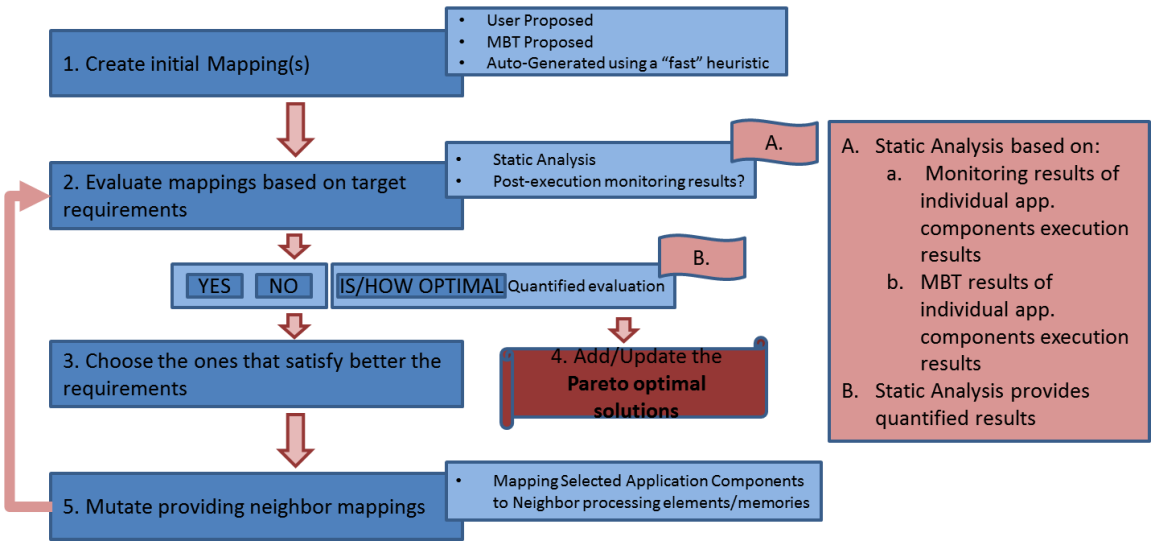


Figure 9: High level representation of Generic MOM algorithm

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 refers to the mutated mappings as well. The current supported requirements are user defined and range from total execution time, 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, energy, etc.). Currently, MOM tool estimates the total computation time, the total communication time and the total execution time, based on the metrics (CPU speed, memory size and memory access time) provided by the Platform Model (hardware platform specification).

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 the application's total execution time.

Step 4: The next step consists of the Mutation of the best mapping, providing neighbor mappings. There are two 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 into two subcomponents, (3) generation of random mappings based on the modified component network. This operation is aiming on increasing the component parallelism in the mutated mappings in order to decrease the total computation time.

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 mapping 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:

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

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:

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

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.

2.2.4 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:

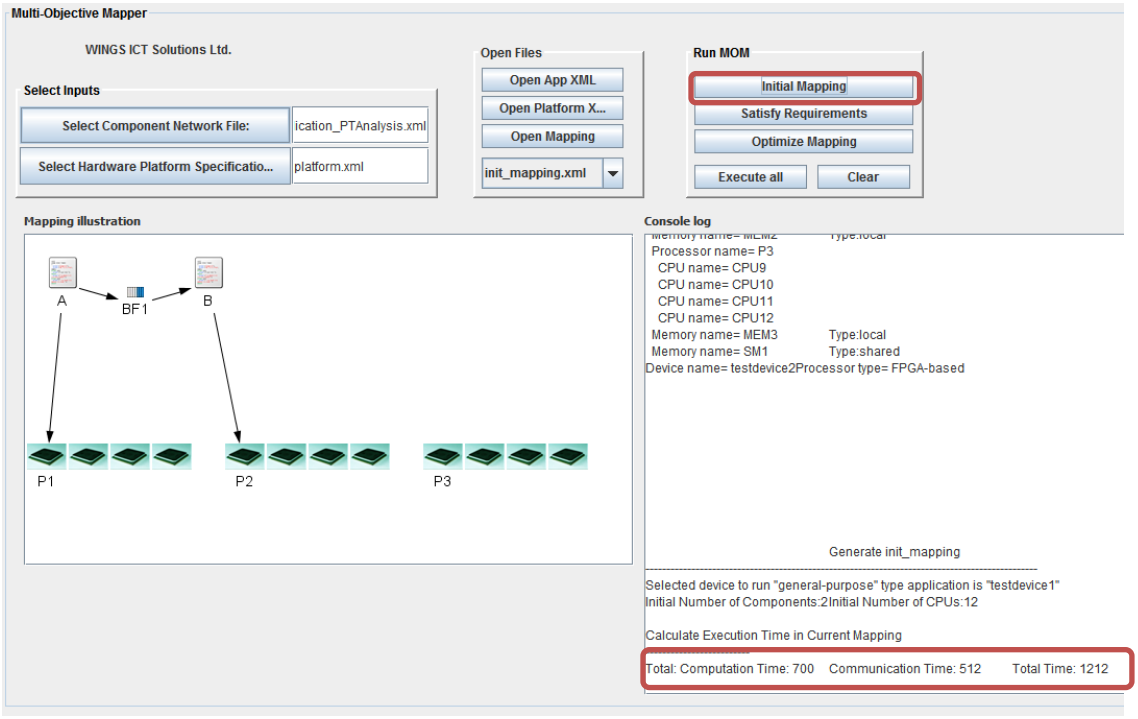


Figure 10: Generic MOM evaluation-GUI along with initial mapping

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".

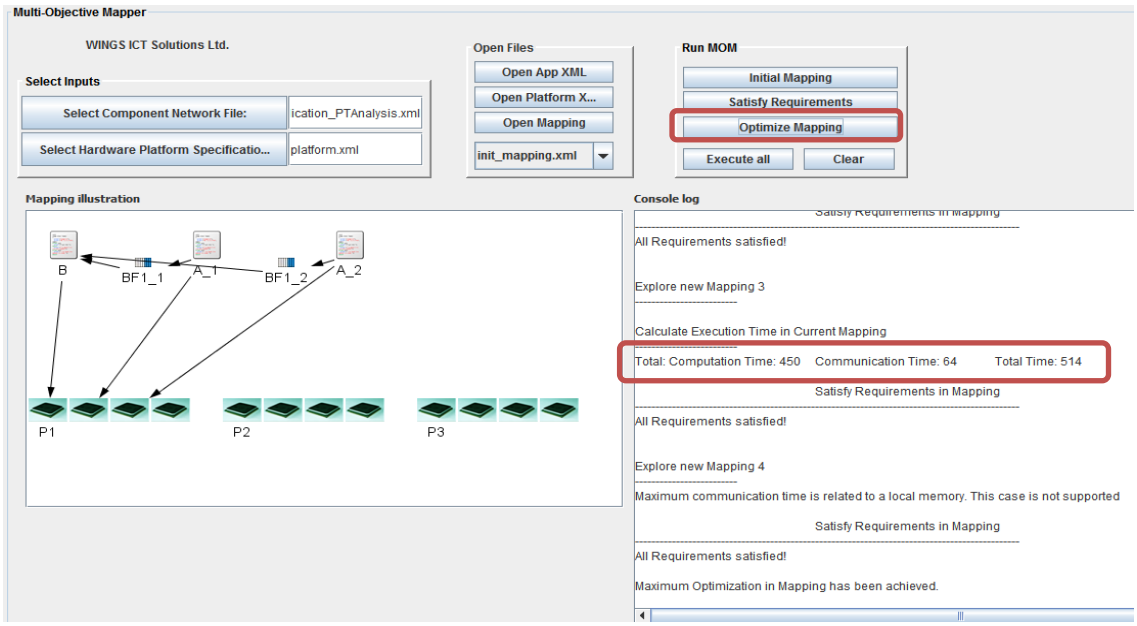


Figure 11: Generic MOM evaluation-Optimized mapping along with results

The above figure depicts the MOM outcome which is the optimized deployment. In this example, MOM split component "A" in 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 provided with the progress of the MOM refinements and the experimentation with the PHANTOM use cases.

2.2.5 Dependencies/integration aspects

In order to provide the best mapping, MOM interacts with a multitude of PHANTOM mechanisms positioned the PHANTOM tool-flow (see Figure 4 and Deliverable D1.2). First, MOM requires the component network, the number of application components and the application components' source code along with the communication objects. These will be provided by the user defined documents that are stored inside the Repository through the Programming model. Similarly, MOM will require the System configuration, the Initial Mapping, the System model along with user/use case requirements from the Repository through the Programming model. Furthermore, a specific MBT-MOM strategy has been designed 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. Finally, MOM will require the parallelization directives from the Parallelization toolset that further assist MOM to decide on the number of component's parallelization.

2.2.6 Innovations beyond the state-of-the-art

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.

2.2.6.1 Background technologies utilised in development

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

2.2.6.2 Summary of new technologies/extensions developed

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 2.2.3 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 2.2.3), which have been implemented thoroughly in the scope of PHANTOM and optimized for addressing PHANTOM requirements.

2.2.6.3 Early/Full Prototypes functionality

Early-First Year Prototype:

The current version of the Generic MOM prototype is covering non-functional requirements focusing on time performance (both computation and communication) as well as memory utilization. Data movement is considered by the minimization of the communication time. The platforms currently supported in full extent are CPU based architectures. Functional requirements are fulfilled by the execution of the application as defined by the user. All three use cases are currently supported.

Full Prototype and Next Steps:

The next steps of the work include the refinements of the implemented first version of the Generic MOM based on evaluation of MOM in PHANTOM use cases.

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

- Power and thermal, given that there are means to be monitored for the specific processing element from the Monitoring Framework.
- Dependability, by taking into account the outcome provided by MBT functional and/or non-functional testing.
- Security, by taking as input the component network as derived by the application of the Component Network Execution Integrity (see 4.1).

The next version of the Generic MOM will support in full extent CPU-GPU based architectures (development is currently at the final stage, namely testing under the PHANTOM use cases) and CPU-FPGA based architectures, as defined in the problem formulation described in Section 2.2.2.1 (equation (5)). Apart from use case integration, the Generic MOM will also be integrated with the Offline MOM (see 2.1) and MBT-MOM strategy implementation (see 5.2.1.4).

3. RUNTIME MONITORING

3.1 MONITORING LIBRARY

3.1.1 Use case requirements

The run-time monitoring framework plays an essential role for the application optimization based on the understanding of both software non-functional properties and hardware quality attributes with regards to performance and energy consumption aspects. The PHANTOM monitoring library will be built upon the ATOM framework, developed for EXCESS and DreamCloud projects. Below we summarize and discuss main requirements to the monitoring framework, derived from the application specification in D1.1.

3.1.1.1 Heterogeneous target platform (U73)

The PHANTOM monitoring framework should support all mandatory target platforms of the users' interest. To be specific, the PHANTOM infrastructure should be heterogeneous, including multi-core CPUs, GPUs, FPGAs, and the targeted embedded systems (e.g. Movidius). Subject to the hardware facilities and availabilities, sometimes a hosting hardware is required in order to monitor the connected accelerator. For instance, collecting the run-time metrics of a GPU is done by the monitoring framework deployed on the associated hosting CPU. The reconfigurable (FPGA) and hybrid (CPU+FPGA) device monitoring should happen via industry-standard FPGA Mezzanine Connectors (FMC), e.g. Xilinx Zynq platform.

Req. No.	Requirement	Overall Priority
U73	The run-time monitor shall be capable of acquiring monitoring data in all mandatory target platforms (e.g. CPU, FPGA, etc) subject to available hardware capabilities	SHALL

3.1.1.2 Metrics (U26, U72, U74-78, U83)

Generally, the PHANTOM run-time monitoring should support metrics covering both hardware (infrastructure-level) and software (application-level) properties. Some metrics are predefined, like the execution time, memory properties, power consumption, communication bandwidth, and I/O usage, while the others are application-specific metrics, which are user-defined and application-distinctive. Some examples of the user-defined metrics are the number of the processed frames for the surveillance use case or the number of the numerical integration steps for the HPC application. These predefined metrics are also different based on different hardware and sensors availabilities. Table 1 to Table 4 list the predefined metrics for the major platforms (cf. D4.2) according to the users' requirements and hardware availabilities. As Linux constitutes the major OS of the targeted hardware architectures (cf. D4.2), its monitoring services are largely leveraged by the PHANTOM monitoring services.

Req. No.	Requirement	Overall Priority
U26	PHANTOM shall provide an API for monitoring of user-defined metrics	SHALL
U72	The PHANTOM run-time monitor shall be able to monitor non-functional properties of an application	SHALL
U74	The PHANTOM framework should be capable of monitoring execution time properties	SHOULD
U75	The PHANTOM framework should be capable of monitoring memory properties	SHOULD
U76	The PHANTOM framework should be capable of monitoring power consumption properties	SHOULD
U77	The PHANTOM framework should be capable of monitoring communications bandwidth properties	SHOULD
U78	The PHANTOM framework should be capable of monitoring I/O properties	SHOULD
U83	PHANTOM should provide monitoring of application-specific performance metrics	SHOULD

3.1.1.3 Accessibility (U35, U79, U80-U82)

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.

Req. No.	Requirement	Overall Priority
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
U79	The data obtained by the run-time monitor shall be accessible and exposed to the user for their own tasks	SHALL
U80	The user shall be able to have fine-grained control over execution time monitoring by indicating application subcomponents (i.e. tasks, loop, code blocks) whose execution time shall be monitored	SHALL
U81	It should be possible to export the run-time monitoring	SHOULD

Req. No.	Requirement	Overall Priority
	data in a structured data format	
U82	For non-periodic non-functional properties, the user should be able to select the frequency of data acquisition	SHOULD

3.1.1.4 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.

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

Table 1: Predefined metrics for platform with Linux based CPUs

CATEGORIES		METRICS	UNITS
INFRASTRUCTURE-LEVEL	Performance	core[ii]:MFLIPS	Mflip/s
		core[ii]:MFLOPS	Mflop/s
		core[ii]:MIPS	Mip/s
	Resources utilization	CPU_usage_rate	%
		RAM_usage_rate	%
		swap_usage_rate	%
	I/O	io_throughput	Bytes/s
	Network	net_throughput	Bytes/s
	Temperature	CPU[i]:core[ii]	°c
	Power	estimated_CPU_power	milliwatt
		estimated_memory_power	milliwatt
		estimated_disk_power	milliwatt
		estimated_wifi_power	milliwatt
		estimated_total_power	milliwatt
		package[i]:total_power	milliwatt

		package[i]:dram_power	milliwatt
APPLICATION-LEVEL	Performance	execution_time	millisecond
	Resource utilization	CPU_usage_rate	%
		RAM_usage_rate	%
		swap_usage_rate	%
	I/O	disk_read	Bytes
		disk_write	bytes
		disk_throughput	bytes/s
	Power	power_CPU	milliwatt
		power_mem	milliwatt
		power_disk	milliwatt
		power_wifi	milliwatt
		power_total	milliwatt

Table 2: Predefined metrics for platform with Nvidia GPUs

CATEGORIES		METRICS	UNITS
INFRASTRUCTURE-LEVEL	Resources utilization	GPU_usage_rate	%
		mem_usage_rate	%
		mem_allocated	%
	Communication	PCIe_snd_throughput	Bytes/s
	Network	PCIe_rcv_throughput	Bytes/s
	Temperature	GPU_temperature	°c
	Power	GPU_power	milliwatt
APPLICATION-LEVEL	Performance	execution_time	millisecond

Table 3: Predefined metrics for platform with FMC. It is noticed that the following table is based on Xilinx Zynq ZC706 board. As metrics with regards to ARM processors are the same as that for Intel Xeon processor, it is thus not repeated here. The application-level monitoring is the subject for further investigation.

CATEGORIES		METRICS	UNITS
INFRASTRUCTURE-LEVEL	Communication	Read/write transactions	-
		Read/write latency	-
		Read/write throughput	MBytes/s
	Power	ARM_power	milliwatt
		FPGA_power	milliwatt

Table 4: Predefined metrics for Myriad2 embedded system

CATEGORIES		METRICS	UNITS
INFRASTRUCTURE-LEVEL	power	total_power	milliwatt
APPLICATION-LEVEL	power	power_core	milliwatt
		power_ddr	milliwatt
	Temperature	temperature_CSS	°c
		temperature_MSS	°c
		temperature_UPA0	°c
		temperature_UPA1	°c
	Performance	execution_time	millisecond

3.1.2 Design specifications

3.1.2.1 Architecture

As clarified in D1.2, the PHANTOM monitoring framework architecture follows the design of ATOM – the monitoring solution elaborated by the EU projects EXCESS and DreamCloud. 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 12).

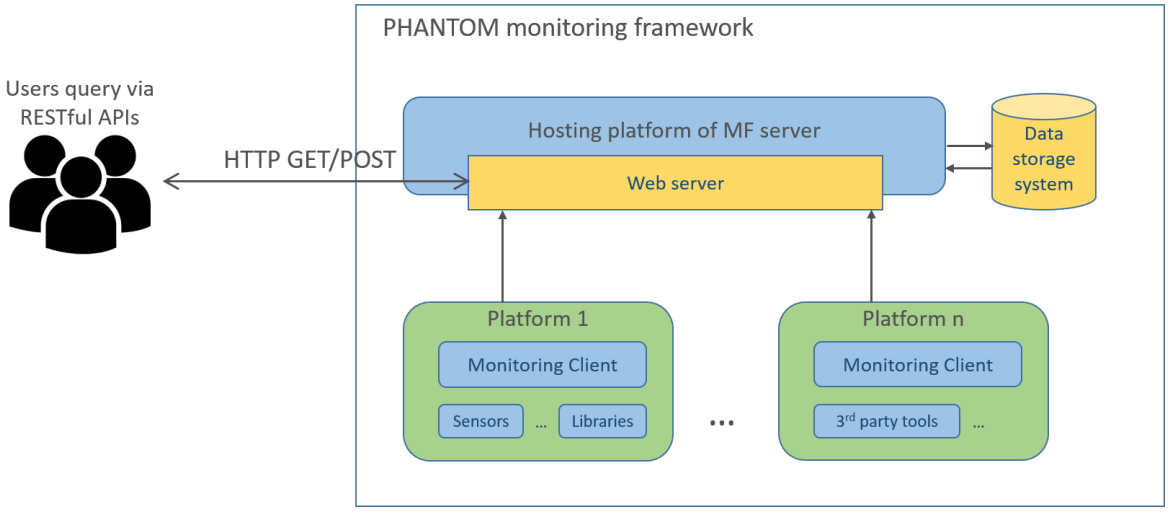


Figure 12: Architecture of PHAMTOM monitoring framework

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.

Before starting the monitoring, a MF client is firstly required to be installed on a specific target platform. Then the MF client collects the user-configured metrics at a customizable frequency and sends the collected data periodically to the server side. Meanwhile, 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.

3.1.2.2 Monitoring client

In order to achieve monitoring of metrics in both software and hardware aspects, the workflow of the MF client is divided into two parts. As shown in Figure 13, users can select to use the monitoring client as an application to monitor infrastructure related metrics, or to use the MF client APIs to enable code instrumentation, or to use these two methods at the same time to obtain both software and hardware metrics.

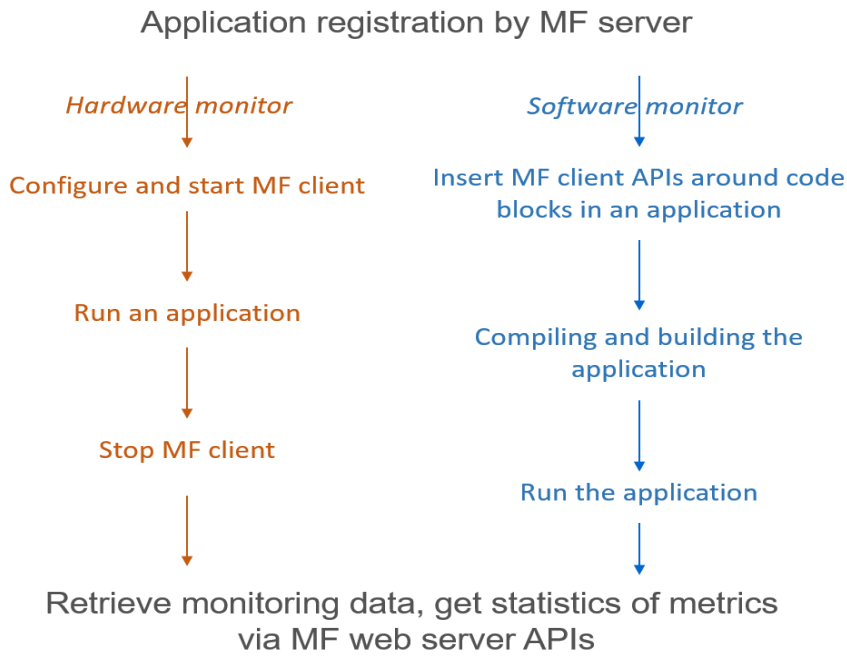


Figure 13: Workflow of application and infrastructure monitoring with MF client

With respect to hardware monitoring, Figure 14 shows the inside workflow of the MF client. On the target platform, the MF client starts with reading and parsing the user-defined configuration file, named as "mf_config.ini". Depending on the configuration of plug-ins and metrics, the plug-in manager checks the availabilities of the user-interested plug-ins, activates the associated metrics and prepares the sampling frequencies. Afterwards, the thread handler creates for each activated plug-in a thread, which is responsible for metrics sampling and metrics publishing of the corresponding plug-in. To be noted, the availability of plug-ins depend on various conditions, e.g. some hardware counters, 3rd party utilities or some libraries. For example, the Intel Xeon processors provide dedicated counters for energy consumption retrieval – RAPL.

Application-level metrics are obtained via using a user library that allows the code instrumentation and fine-grained monitoring. This library is included in the Monitoring Client – the client part of the monitoring framework (see D4.2 for details). In addition to the predefined application-level metrics, the user library supports user-defined metrics profiling as well.

3.1.2.3 Monitoring server

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 15). 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.

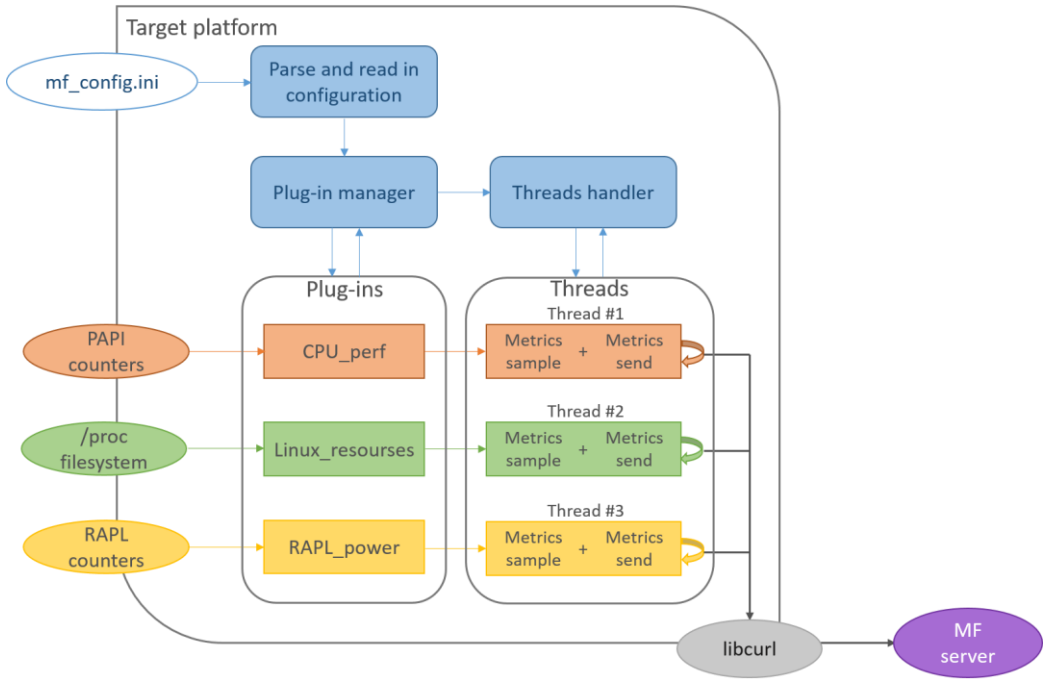


Figure 14: Workflow and components of MF client hardware monitoring

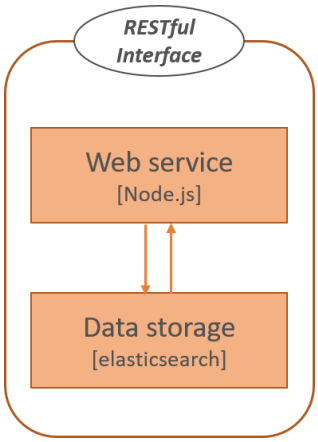


Figure 15: Workflow and components of MF server

3.1.2.4 The communication layer

Same as the ATOM, we keep using JSON as the primary data format for data exchange between the client and the server. As presented in Listing 1, at a specific time point, a simple key-value pair composed of the metric name and its numeric value is sent to the server. 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 case of PHANTOM distributed infrastructure, the timestamps sent are revised to the local timestamps (in millisecond), since the users are mostly interested in rather time duration than the exact real time.

Listing 1: Example a typical JSON metrics

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

3.1.3 Implementation details

The PHANTOM MF client is written in C for the prototype. As Figure 14 shows, the MF client source code is organized accordingly, with each component to fulfil its specific functionality, as can be seen in the following project tree.

```
phantom_monitoring_client/src
├──agent
│  ├──main.c
│  ├──main.h
│  ├──plugin_discover.c
│  ├──plugin_discover.h
```

```
│   ├──plugin_manager.c
│   ├──plugin_manager.h
│   ├──thread_handler.c
│   └──thread_handler.h
├──core
├──parser
│   ├──Makefile
│   └──src
├──publisher
│   ├──Makefile
│   └──src
├──plugins
│   ├──Board_power
│   ├──CPU_perf
│   ├──CPU_temperature
│   ├──Linux_resources
│   ├──Linux_sys_power
│   ├──NVML
│   ├──RAPL_power
│   ├──utils
│   └──README.md
├──api
│   ├──Makefile
│   └──src
└──mf_config.ini
```

The /agent folder implements the management of various plug-ins and threads, playing a role as the main control unit. The folders /core, /parser, and /publisher are used by the main controller for accessorial support, like parsing input configuration file, publishing metrics via HTTP, and so on. The implementation of each plug-in is collected in the folder /plugins, where associated metrics are sampled and formatted periodically. The MF client is designed to be pluggable. Loading a plug-in means starting a thread for the specific plug-in based on the users' configuration at run-time.

Currently, seven plug-ins are supported by the PHANTOM MF client[1]:

- Board_power – monitoring device power consumption by means of an external ACME board
- CPU_perf – information from the PAPI hardware counters of a CPU
- CPU_temperature – temperature data from the dedicate linux monitoring sensors
- Linux_resources – information from the Linux processes and the file system services
- Linux_sys_power – CPU power consumption data, as estimated by the Linux core services
- NVML – information from NVIDIA management library for GPU-accelerators
- RAPL_power – data on CPU power usage from the dedicated RAPL-counters (available only for some Intel XEON CPUs).

Please refer to D4.2 for more details about each plug-in's implementation.

The application-level monitoring and user-defined metrics collection are implemented in the /api folder, which provides a user library and several APIs for application code instrumentation. Some description for the interfaces and their usage introduction are given in Section 3.1.4.1 and the full details are provided in D4.2.

3.1.3.1 Monitoring server

The MF server is written in JavaScript under Node.js runtime environment. Node.js has an event-driven architecture and is qualified 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, is selected for building the RESTful APIs.

For data storage component, Elasticsearch 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.

[1] https://github.com/phantom-monitoring-framework/phantom_monitoring_client/tree/master/src/plugins

3.1.3.2 Source release and GIT repositories

The source code of PHANTOM monitoring framework is released using both the project's SVN and via 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_client" and the "phantom_monitoring_server", which can be accessed via the following links:

https://github.com/hpcfapix/phantom_monitoring_client
https://github.com/hpcfapix/phantom_monitoring_server

3.1.4 Dependencies/integration aspects

3.1.4.1 Application-level APIs

Consumers of the monitoring information (the end-users or the PHANTOM platform services such as MOM) require to have a fine-grained control over execution time monitoring by indicating application sub-components (i.e. tasks, loop, code blocks), we design and enable application-level monitoring via building a user library, as providing programmers necessary simple interfaces. Following interfaces (c.f. Listing 2) are developed and included in our first prototype release. In addition to collecting generic metrics of performance and power, the APIs are capable of monitoring application-specific metrics as well.

Listing 2: PHANTOM MF client APIs

```
/*
Start monitoring of the predefined metrics for sub-components of an application. Data are stored at first locally. Required input parameters should include the MF server URL, name of the platform, where the application runs, and the metrics' name and sampling frequency.
*/
char *mf_start(char *server, char *platform_id, metrics *m);

/*
Stop monitoring of the predefined metrics when the sub-component is finished.
*/
void mf_end(void);

/*
Send locally-stored predefined metrics to the PHANTOM MF server. The unique generated execution ID will be returned on success.
*/
char *mf_send(char *server, char *application_id, char *component_id, char *platform_id);

/*
Send user-defined metrics with given metric's name, value, and current local timestamps to the PHANTOM MF server. Returns the status of the operation completion.
*/
int mf_user_metric(char *metric_name, char *value);
```

3.1.4.2 RESTful APIs

The following functionalities and services are supported by the PHANTOM MF server RESTful APIs (c.f. Table 5):

- MF client uses the RESTful APIs for sending sampled metrics and timestamps.

- It provides means for users 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 can obtain a fine-grained execution time of a sub-components of an application by specifying the unique generated execution ID.

- Simple statistics and data analytics are fulfilled by query specific RESTful APIs.

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

Table 5: RESTful APIs of PHANTOM MF server

workflows		
/workflows	GET	Get all applications registered by the PHANTOM MF server
/workflows/:application_id	GET	Get a specific application details, such as an id.
	PUT	Register a new application by the PHANTOM MF server
experiments		
/experiments	GET	Get all available experiments
/experiments/:execution_id	GET	Get a specific experiment details, such as an id.
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
runtime		
/runtime/:application_id/:task_id/:execution_id	GET	Get the execution time of a specific experiment
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

3.1.5 Innovations beyond the state-of-the-art

The following major innovations are identified for the Monitoring Library:

- **Monitoring of user-defined metrics for the application-level monitoring.** Without the need of installing any additional monitoring tools, application (or user) specific metrics can be monitored. For this purpose, the Monitoring Library offers the users a set of light-weight, hardware-agnostic API for injecting the instrumentation into the application codes.

- **Highly customizable monitoring settings.** The PHANTOM monitoring framework was designed with the user-friendliness in mind – the users can specify the metrics to be automatically collected, the flush time interval, configuration of collected metrics, etc., even at run-time, which is not the case for the other, alternative approaches.

- **Integration of the application-specific monitoring data into visualization back-ends**. In addition to tools like *Grafana*[2] that can easily be used for the infrastructure-specific monitoring visualization, the PHANTOM Monitoring Library allows an easy extension of the visualization tools to tackle with the application-specific metrics, stored in the PHANTOM Monitoring Database.

3.1.5.1 Background technologies utilised in development

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

Therefore, the user's extensions of the Monitoring Framework – the Monitoring Library – were largely developed from scratch (with the reuse of the design outcomes of the EXCESS[3] project). The elaborated API syntax follows the one used in the well-established tools for parallel applications profiling like Extrae[4] and Vampir-Trace[5] and is tightly integrated with the Monitoring Client functionality (cf. D4.2).

[2] https://grafana.com/
[3] http://www.excess-project.eu/
[4] https://tools.bsc.es/extrae
[5] https://tu-dresden.de/zih/forschung/projekte/vampirtrace

3.1.5.2 Summary of new technologies/extensions developed

The PHANTOM Monitoring Library provides a way to monitor **user-defined metrics –** any application-specific properties that are of interest for the user (the application developer). For instance, a user-defined metric can be the number of numerical equations solved by the HPC use case or the number of frames processed from a video stream by the Surveillance user case. The users can thus analyze the changes in the collected metrics across multiple execution 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.

The use of the Monitoring Library needs to be as simple and intuitive for the user as possible. The listing below shows a simple example of using the Monitoring Library for defining and gathering user-specific metrics.

```
int main(){
    ….
    int nr_of_steps = 0;

    while (…) {
        …
        nr_of_steps++;
    }

    string my_new_metric_name = “Nr_of_steps_performed”;
    mf_user_metric(my_new_metric_name, nr_of_steps); // register of a user metric
    ….
    monitoring_send( ); //sends the buffered monitored data, including the user metrics, to the server
    ….
}
```

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

The Monitoring Library also provides a very **flexible way to configure** all default metrics that should be collected for the application by the Monitoring Client. For this purpose, the function *mf_start* was extended to accept the list of plug-ins that need to be activated for each specific application run. The following listing demonstrates how the list of metrics to be monitored as well as sampling intervals for them can be defined.

```
                    /* MONITORING METRICS */
    Phantom_mf_metrics m_resources;
    m_resources.num_metrics = 2;
    m_resources.local_data_storage = 1; /*remove the file if user unset keep_local_data_flag */
    m_resources.sampling_interval[0] = 1000; // 1s
    strcpy(m_resources.metrics_names[0], "resources_usage");
    m_resources.sampling_interval[1] = 1000; // 1s
    strcpy(m_resources.metrics_names[1], "disk_io");
    m_resources.sampling_interval[2] = 1000; // 1s
    strcpy(m_resources.metrics_names[2], "power");
    /* MONITORING START */
    mf_start(server, regplatformid, &m_resources);

    …
```

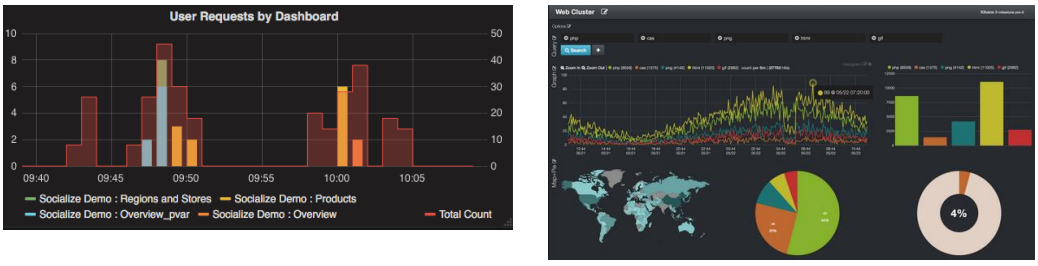
Integration with command-line visualization back-ends. Unlike the commodity and HPC systems, the embedded and low-power devices cannot perform visualization with compute-intensive tools like Grafana and Kibana. For this, a set of simple client GUIs was developed to enable visualization through the command line (in a textual form).

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 revealing the application from the need to exploit any additional visualization tools. The commodity tools use those visualization back-ends for infrastructure data only.

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 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. However, that data need some pre-processing when we wish to cross data from different types of data such video frames processed per watt, find relevant frames of data among the data stored, or other results from data analytics.



(a) (b)

Figure 16: Example of plots with (a) Grafana and (b) Kibana.

3.1.5.3 Early/Full Prototypes functionality

Early-First Year Prototype:

- Schema for the application-specific events tracking by means of the Monitoring Database.
- Integration with the Monitoring Client for user-specific metrics data transfer to the Monitoring Server.
- Basic integration with the major visualization GUIs.

Full Prototype and Next Steps:

- Monitoring Library completed.
- Extended API for tackling with parallelized applications.
- Extended integration with visualization back-ends.
- Data analytics and visualization tool for supporting multi-dimensional optimization.

4. SECURITY

There are many extant approaches and mechanisms to address the many layers and aspects of security in a variety of system contexts. In Section 9 of PHANTOM deliverable D1.2 we described our strategy to identify the security problems that we should address with the limited resources dedicated to that purpose in this project. Our approach is to not reinvent the wheel, but to focus on the security problems that are uniquely caused or exacerbated by PHANTOM's specific innovations, and for which these concerns cannot be adequately addressed by an existing security feature or security product. PHANTOM's specific innovation is that it aims to provide seamless integration and orchestration of heterogeneous computing elements while exercising control over the qualities exhibited by their combined operation over a broad range of scale.

Accordingly, we identified two general security problems linked to PHANTOM's specific innovation:

- ***execution integrity*** of a PHANTOM application's component network, and
- coherent, uniform and flexible ***access control*** over resources for large-scale PHANTOM applications within a distributed and heterogeneous operating environment.

Both of these two problems exist at different levels in the PHANTOM platform. Concepts associated with these two security problems, and rationale for focusing on these two, were presented in PHANTOM deliverable D1.2, Section 9. A brief summary of each is presented in the following sections.

4.1 COMPONENT NETWORK EXECUTION INTEGRITY

A prerequisite of security, as well as of any predictable behaviour of a computation, is the integrity of execution and of resources. If the integrity of memories, registers, data transfers, and instruction execution cannot first be guaranteed, then the suitability of a system's architecture and the correctness of its logic for its purpose are inadequate to guarantee useful system properties. For example, if the contents of memories and

registers cannot be isolated so that only a known process can modify them, and if the sequence of instruction execution in such a process is subject to unforeseen perturbation by activities outside of the process, then we cannot predict the effects of a computation.

Application execution integrity is a precondition for confidence that the application will function as expected. Specific security mechanisms, in addition to operational application logic, may be appropriate to a particular application. Like the operational components they complement, security mechanisms cannot be relied upon if execution integrity cannot be guaranteed.

Integrity of execution is based on Isolation of resources and Information Flow Control among them (IIFC). Resources are used to create subjects (processes) and objects, which are in turn allocated to components. Interactions among components are constrained by a component network to those that are needed to achieve the purpose of an application. The component network is created by mechanisms that limit the interactions.

PHANTOM must provide integrity of execution of component networks, which are the organizational model for PHANTOM applications. The foundation for integrity of execution is IIFC provided by the execution environment. Analysis of the design of the execution environment provides confidence in the environment's ability to provide integrity of execution. The architectural design of the software and/or firmware that works with each hardware execution component (CPU, GPU, and FPGA) must justify how IIFC is achieved for that component and between execution components and memories. The Open Group will work collaboratively with the respective designers to examine these architectural designs for adequacy.

4.1.1 Use case requirements

Every use case implicitly expects execution integrity. In D1.1, section 5.8, we have referred to relevant System and Data Security requirements derived by the PHANTOM use cases, e.g. U66. However, execution integrity is a fundamental assumption that every programmer makes. Thus, besides being an explicit requirement of any application, it is an implicit requirement of every application.

Req. No.	Requirement	Overall Priority
U66	PHANTOM should support means for tasks isolation and information flow control policy	SHOULD

4.1.2 Design specifications

Component network execution integrity was introduced in PHANTOM deliverable D1.2, Section 9.5. Global execution integrity of an application component network is not a "function" provided by some "module" of the execution platform. Particularly in a distributed and heterogeneous platform, it is an emergent property of the platform components in combination, the result of the composition of those components' behaviors.

But what behaviors are needed of each component to achieve this property of the platform? It is just this question that drove the development of the MILS[6] platform. The components of the MILS platform are specified in such a way that their composition exhibits IIFC and integrity of execution integrity of application components that run on the MILS platform [11]. PHANTOM adopts the approach exemplified by the MILS platform but without particular preexisting platform component specifications, and without the extreme rigor of high assurance that is a hallmark of many MILS applications.

The MILS approach defines the MILS platform, as illustrated in Figure 17, as a composition of resource managers for different resource types that together achieve coherent IIFC. What is required of the PHANTOM platform is a persuasive case that the composition of the PHANTOM platform components delivers the coherent IIFC needed for predictable execution of PHANTOM application component networks.

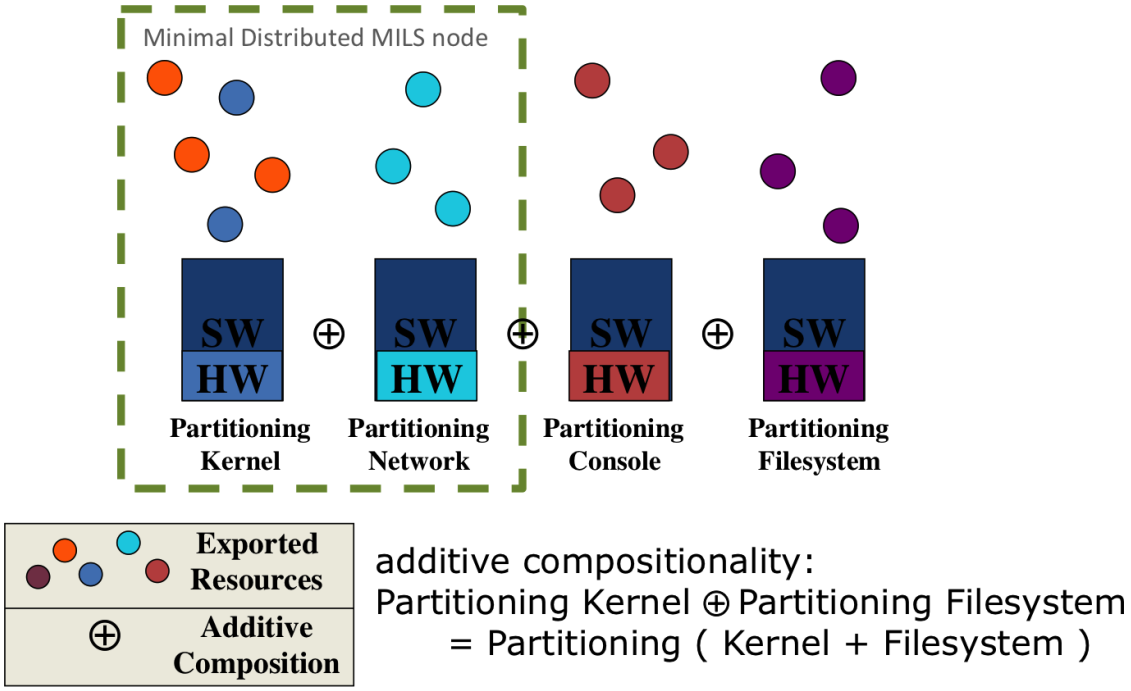


Figure 17: MILS platform component composition

Some of the resource managers may have multiple instances in a distributed environment. Each manager locally provides IIFC for the resources it manages and the instances of the type it creates. It also composes with other managers of the same type in the distributed system to act as one distributed manager for the type. Managers of different types in turn compose to maintain the isolation properties of the individual managers but to provide coherent information flow control among diverse types.

Each type of PHANTOM heterogeneous compute elements is to be paired with control software or firmware to make it conform to a compositional model like that of MILS

[6] Originally MILS was an acronym for "Multiple Independent Levels of Security". This name was found to not be especially accurate. Today, we retain "MILS" not as an acronym but as a proper name for an architectural approach and an implementation platform supporting that approach.

and achieve coherent IIFC. Such a composition of PHANTOM foundational components would support a PHANTOM application component network as shown in Figure 18.

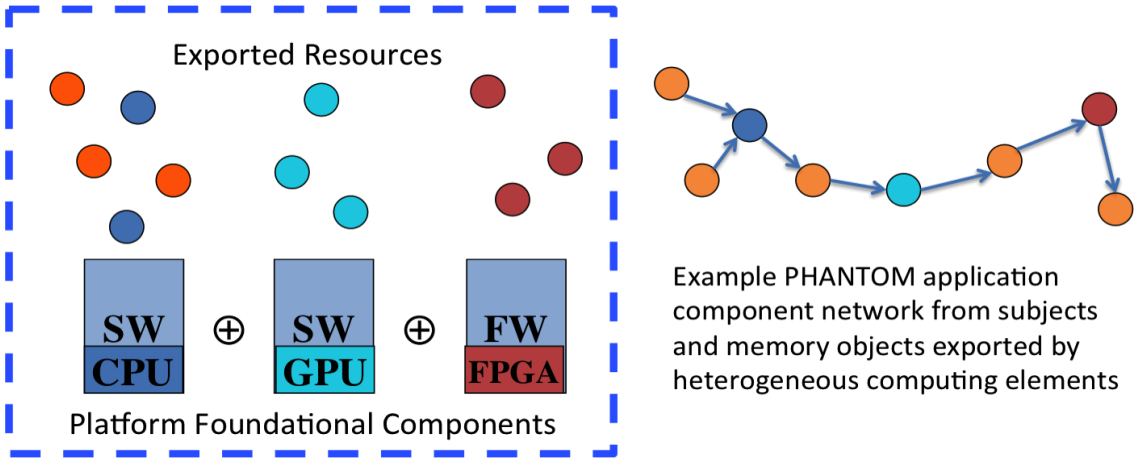


Figure 18: Foundational compute components of PHANTOM platform

4.1.3 Implementation details

IIFC is to be provided in both the PHANTOM development platform and the deployment platform. IIFC is provided in the PHANTOM development platform by the operating environment in which the tools of the PHANTOM system are run. For example, language- or operating system-provided facilities for resource and process isolation will be used to provide integrity of execution of the tools and subsystems of the PHANTOM architecture in the application development phase.

IIFC is provided in the PHANTOM deployment environment by a combination of mechanisms in the operating environment, by the deployment manager, and by software and/or firmware implementations that work in conjunction with the hardware elements of the platform to form MILS-style foundational components that individually provide IIFC and compose to deliver global IIFC for application runtime execution integrity. The implementation of each software/firmware/hardware foundational component must embody a IIFC strategy for resource management that integrates with the strategies of other components to deliver global IIFC and application execution integrity.

The resource managers referred to are the runtime managers of the physical computational resources (processors and memories) that permit safe sharing of those resources. These are the platform foundational components depicted in Figure 18, each the combination of its hardware and its controlling software and/or firmware. For example the combination of an operating system kernel and a CPU enables the CPU to be safely shared among multiple processes.

4.1.4 Testing results

“Testing” for integrity of execution is not accomplished by conventional test development and execution but by architectural inspection and analysis. This is because one would be attempting to test a negative proposition, that is, for example, that the memory of a process cannot be read or written by another process unless an explicit

shared memory is intended (and in that case what is to be established is that no third process can read or write the memory shared by the first two processes). Test cases for such properties tend to be difficult to develop and run because if the developer knows of a test that is expected to succeed then the result is already known and the desired property is not achieved. Further if the developer cannot think of a test to write, then the property ***may*** hold.

In such cases, the traditional approach is to provide an argument, from an architectural point of view, why the property is believed to hold. This is a justification from architectural analysis. Such an approach applies to properties such as infiltration (writing a memory without authorization), exfiltration (reading a memory without authorization), and mediation (causing, without authorization, unintended reading/writing of a memory by the processes that are authorized to read/write it).

In PHANTOM, integrity of execution can be characterized by the following architecturally justified properties:

That for each kind of processing element, process isolation is provided during the execution of a process, whether by temporally partitioned exclusive use of the processing element with appropriate sanitization before or after each execution period; or by logical partitioning afforded by a mechanism that provides non-interfering execution of multiple processes simultaneously.

That for each kind of data storage element, operations on that storage element are mediated in accordance with the principle of noninterference of processes and the policy of non-interference implied by the component network of the application.

The testing of these properties will be carried out by inspection of the specifications and implementation of the PHANTOM architecture and of its components.

4.1.5 Dependencies/integration aspects

This activity entails design of the components of the PHANTOM architecture and platform to provide adequate IIFC features, and providing evidence of its adequacy in an architectural design from the standpoint of global IIFC to provide an argument for execution integrity. This design activity will be presented in D1.3 as a whole, but also wherever the design and implementation details of the components of the PHANTOM architecture are presented. It is dependent upon access to adequate documentation or implementation artefacts to complete this analysis. Feedback will be provided to developers of PHANTOM components when adequate documentation or evidence is lacking to complete the analysis or when such documentation reveals that the components do not collectively justify a claim of execution integrity.

4.2 FLEXIBLE AND UNIFORM DISTRIBUTED ACCESS CONTROL

A *reference monitor* is an access control concept describing an abstract machine that mediates all references by any programme to any programme, data or device according to a well-defined policy that specifies the authorized types of reference based on user

and/or programme function. Figure 19 illustrates the reference monitor concept. The reference monitor mediates access by subjects (active entities) to objects (passive entities) according to a defined security policy. As the figure suggests, the reference monitor must be non-bypassable, that is, it is always invoked when a subject operates on an object. Furthermore, it must be tamper-proof, for otherwise it could be tampered to change the policy or to not enforce the policy.

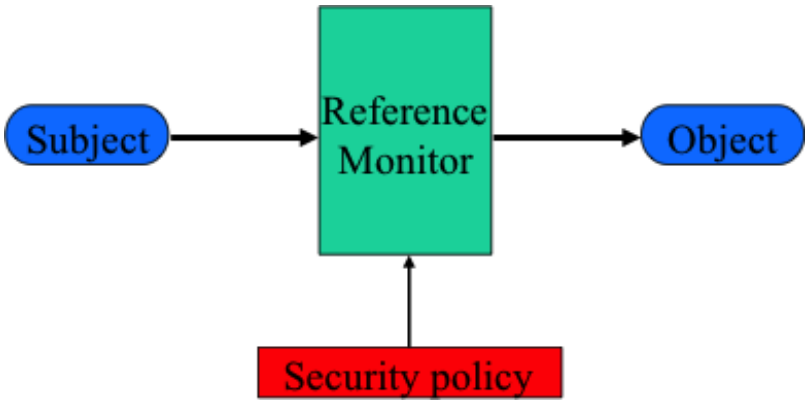


Figure 19: Reference Monitor Concept

The reference monitor concept is valid in a multitude of implementations and is typically illustrated as a monolithic unit with only the security policy portion depicted as being outside. Historically, implementations of the concept as a *reference validation mechanism* (RVM) have been largely monolithic and centralized. Because PHANTOM is implemented in a potentially distributed and heterogeneous computing paradigm, its realization of the reference monitor must also be implemented in a fashion that can serve distribution and heterogeneity.[7]

4.2.1 Use case requirements

In D1.1, section 5.8, we have referred to relevant System and Data Security requirements derived by the PHANTOM use cases, e.g. U67.

Req. No.	Requirement	Overall Priority
U67	PHANTOM should be able to support HPC/Cloud security mechanisms to protect data and control data access	SHOULD

4.2.2 Design specifications

Flexibility in the implementation and deployment of the reference monitor concept, including adaptation to the needs of distributed applications and heterogeneous resources is achieved by a decomposition of the reference monitor into a functional architecture, illustrated in Figure 20, which has been established as a standard known as Next Generation Access Control (NGAC). [10][14][15]

[7] We will refer to the instantiation of the reference monitor concept for PHANTOM as a distributed reference monitor (DRM) that can provide access control over heterogeneous subjects and objects (collectively, resources).

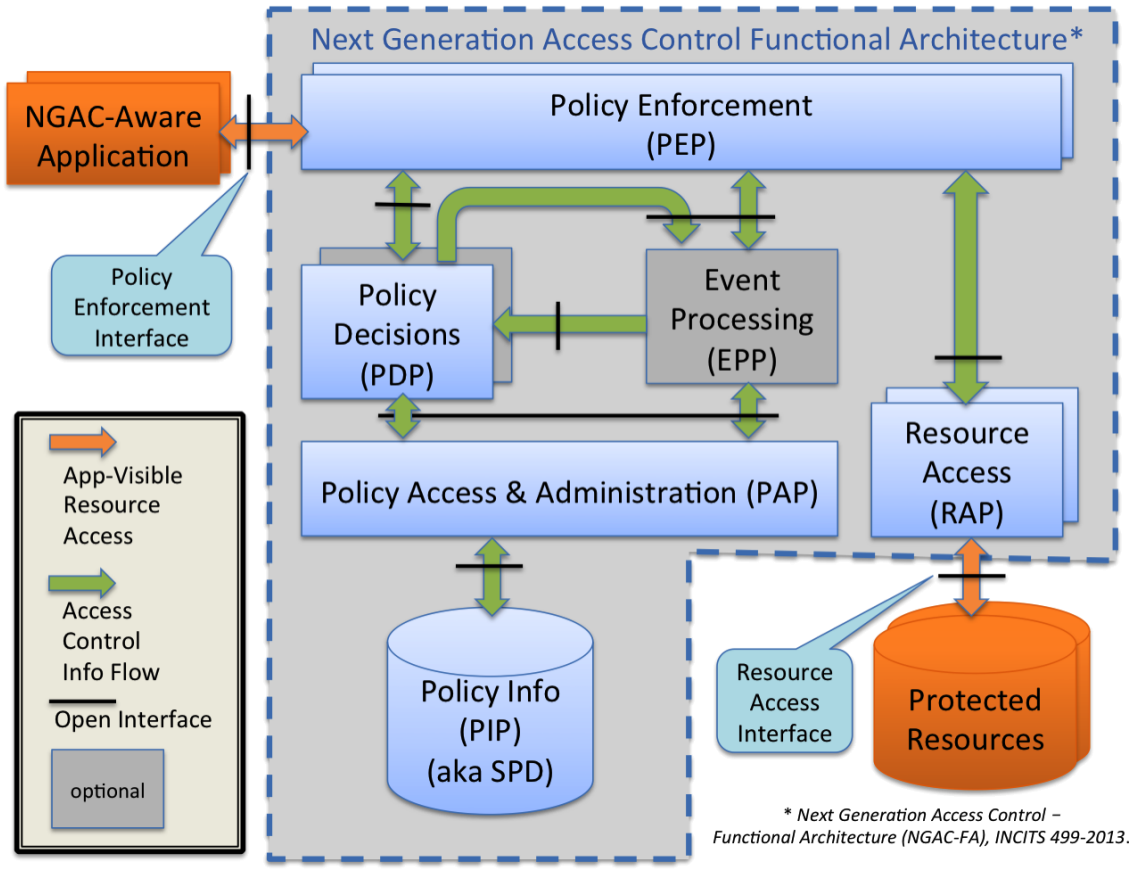


Figure 20: Reference Monitor Functional Architecture

All of Reference Monitor Functional Architecture is at runtime. It interacts with the general Tool flow presented in D1.2, so as the latter to create the policy data that is maintained in the PIP. The NGAC-aware apps are enterprise class PHANTOM apps, and they use the policy enforcement interface (API) to request access to Protected Resources. Without the reference monitor, the PHANTOM Application would directly access the Resource using whatever interface the resource provides. When we interpose the reference monitor the resources become "protected" by the RM. Because the apps and resources may be distributed throughout the system, the parts of the RM are also distributed. More specifically, for the reference monitor to support distributed subjects and objects, the components of the functional architecture may be distributed in various ways across the PHANTOM platform, hence we refer to it a *distributed reference monitor* (DRM for short).

The Open Group's implementation of NGAC for PHANTOM is inspired by the NGAC standard and extends the NGAC reference implementation, known as the "Policy Machine" (PM). The main components of the PM reference implementation are shown in Figure 21. The bullets inset in the figure identify the main components and their roles. PM-aware applications interface with a "node manager" that consists of a Kernel Simulator and a Session Manager. The node manager interacts with the PM Server (policy decision point) to obtain verdicts on access actions requested by the application. The Server consults the Security Policy Database (policy information point) to reach these verdicts. The node manager (policy enforcement point) is trusted to enforce the

verdict. A PM Admin Tool is used to administer the Security Policy DB. All communication among PM components is over the "PM Bus" which is implemented as SSL sockets.

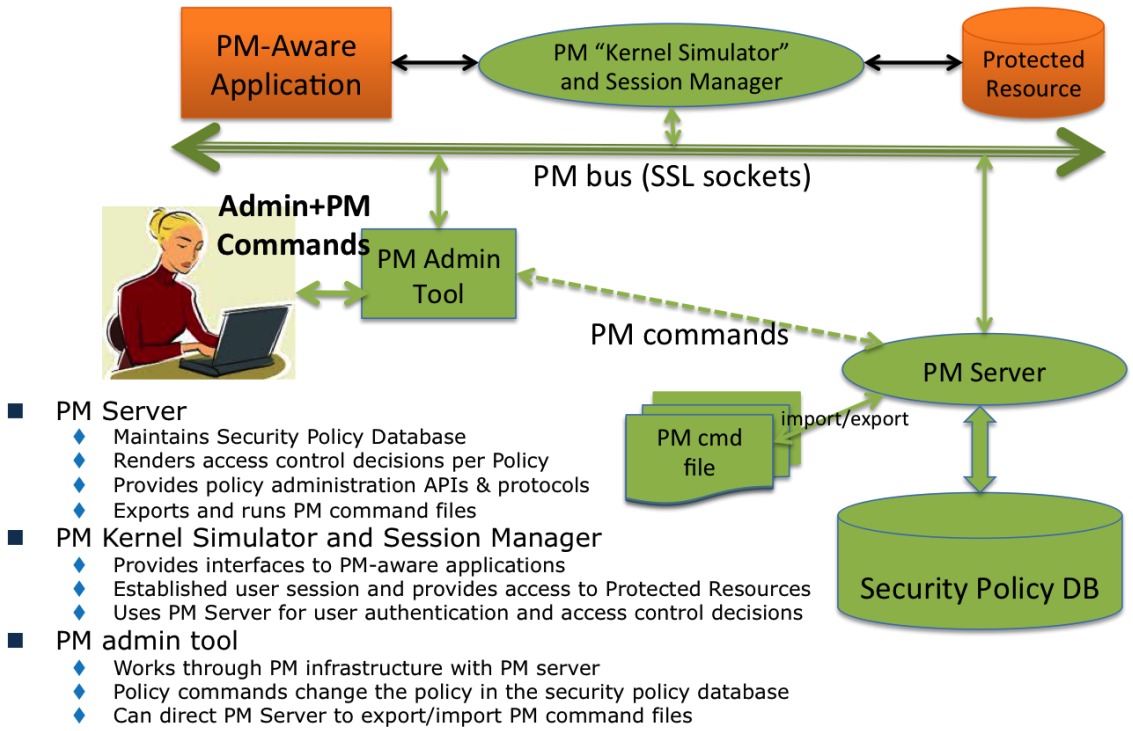


Figure 21: NGAC Reference Implementation (PM)

The NGAC standard includes a policy description framework that is very expressive. Common classes of policies, such as role-based access control (RBAC), discretionary access control (DAC), mandatory access control (MAC), and others can be expressed within NGAC's attribute-based access control (ABAC) scheme. [13] Multiple policies may be in force simultaneously. However the PM implementation is heavyweight in nature, requiring much infrastructure to run, including a relational database management system for storing the security policy database.[8] Consequently, it may be suitable for "enterprise class" PHANTOM applications, where support is possible for the extensive infrastructure required, but not for PHANTOM embedded applications and small target platforms.

Two adaptations of the NGAC approach are being investigated in PHANTOM. The first of these, for distributed enterprise class (EC) PHANTOM applications, requires the PM implementation to be extended for PHANTOM data resources, heterogeneous processing elements, heterogeneous operating environments, and process structures being employed, e.g. micro-services, Web services, and interfacing with the PHANTOM deployment manager. This scenario extends the existing reference implementation and entails schemes for new resource types, and porting of the "node

[8] This is an improvement over a previous version of the reference implementation that required a Microsoft Windows Server with LDAP, Active Directory (AD), and Certificate Server.

manager" (the name we use to refer to the PM Kernel Simulator and the Session Manager collectively) to non-Windows hosts, viz. Linux for PHANTOM.

The second adaptation of the NGAC approach being explored is support for lower level access control policy support for embedded systems. For embedded applications there are static and dynamic deployment possibilities. For statically configured access control the deployment manager would consult the policy server to validate permitted resource use and configure the deployed application and resources to provide only permitted accesses. There would not be a runtime component of the NGAC system, only controls in the operating environment. A dynamic access control scenario would include the simple server and node manager in the PHANTOM runtime system and applications would have to be written to access resources through the runtime system.

The dynamic scenario entails development of experimental lightweight versions of the NGAC policy server and node managers for smaller platforms and their operating environments. These would be less demanding of resources and supporting infrastructure than is the NGAC reference implementation. The goal would be the ability to write policies for low level resources and enable these policies to be queried through a lightweight policy decision point, either at deployment time or at run time, by ad hoc policy enforcement points placed within the PHANTOM platform architecture and/or the deployed application's component network.

The two aspects of NGAC evolution for PHANTOM are illustrated in Figure 22. The Open Group is seeking to extend the basic components and functions of the NGAC reference implementation PM. The first extension is a lightweight standalone policy tool called 'ngac'. This tool introduces a declarative language for expressing attribute-based policies and converts this input to an internal policy model. The tool allows the user to query the policy model, aiding the development and testing of new policies. The tool also permits the conversion of policies from the declarative language form to an imperative language form that is understood by the PM Server. This first extension was developed on the ProSEco Project. [12] The second contemplated extension is the development of a simpler and (more) portable NGAC server and node manager.

For the PHANTOM project there are some necessary extensions that are needed if the PM reference implementation is to be used: managers for resource types required by PHANTOM but not currently supported by PM, namely web services and micro-services for enterprise class PHANTOM applications, interfaces for PHANTOM tool components, and possibly interfaces for use by low-level embedded components. In addition to these extensions an investigation is underway to determine the extent to which development of the simple server/node manager might be undertaken in PHANTOM.

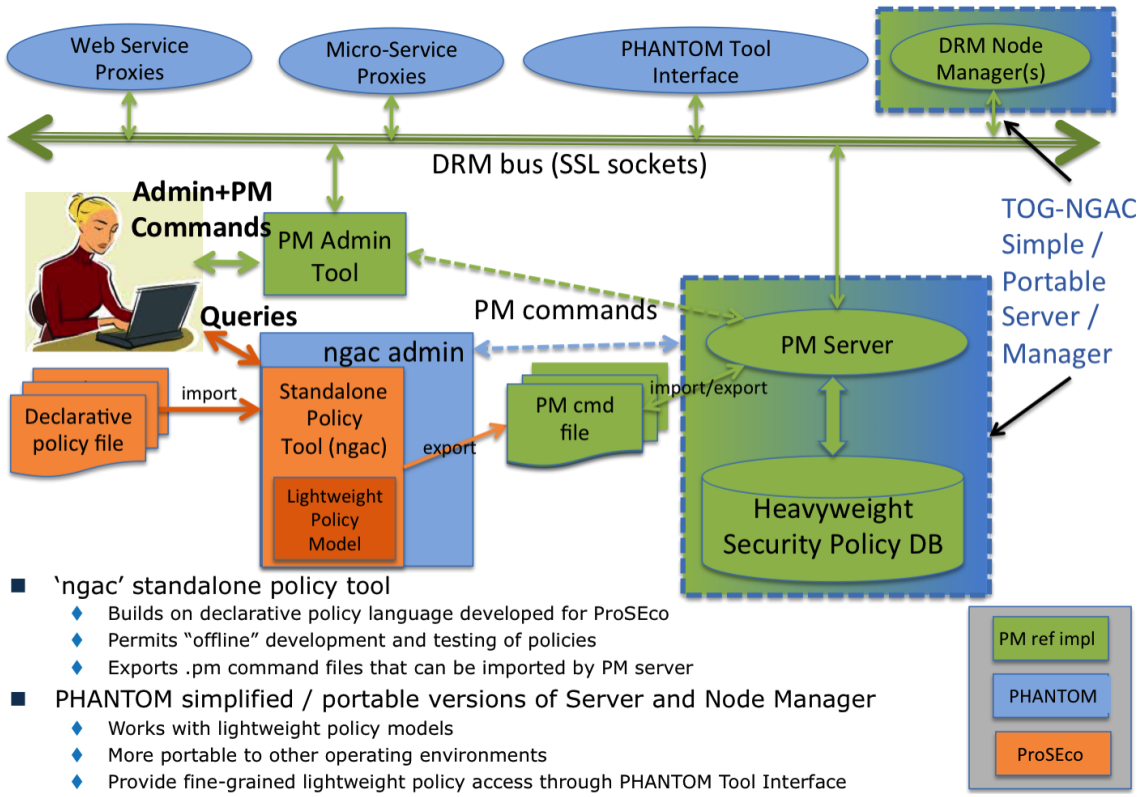


Figure 22: NGAC evolution for PHANTOM

The lightweight policy server would utilize the declarative policy language developed for the ProSEco Project [12] as the basis for the frontend for the lightweight policy server. Interfaces to the policy server would be provided through the PHANTOM Tool Interface, permitting PHANTOM tools such as the MoM or the Deployment Manager to make configuration- or deployment-time queries of the policy or components of the application's component network to make queries at runtime. At the time of this report we are investigating the feasibility of implementing the embedded NGAC adaptation within the resources of the PHANTOM project.

The declarative policy language must be extended to support new object types for resource kinds, web-services and micro-services, not currently supported by the reference implementation or the ngac standalone policy tool.

4.2.3 Implementation details

The reference implementation of the NGAC standard is a system of components, the primary ones being the PM server, the PM admin tool, the Kernel Simulator, and the Session Manager. These components are implemented in Java and communicate over SSL sockets. It was developed to run on Windows Server, with Lightweight Directory Access Protocol (LDAP), Active Directory (AD), Certificate Authority, and other system and network services. Though a new version was recently released that eliminates the need for Windows Server, LDAP and AD, considerable support is still required from the IT environment in which the PM operates. In place of LDAP/AD the PM reference implementation now uses a MySQL database, which in turn has demands.

A prototype of a lightweight standalone tool (ngac policy tool) for the specification and testing of attribute-based access control policies has been developed. It supports a declarative policy language that is more intuitive than the command language of the NGAC PM reference implementation. The ngac policy tool reads a policy specification and constructs a lightweight internal model. From this model it is able to compute derived privileges, a logical object space that is visible to a given user, and other consequences of the specified policy. This can be used to develop and test a policy in a standalone way. The tool is implemented in the Prolog language, which is particularly well suited to the symbolic manipulations and graph computations that characterize this task.

The query and test functionality of the ngac policy tool form the core of what could be a new simplified policy server mentioned previously, though to complete such a server would involve creating the secure communications framework for the DRM bus if it is to be used dynamically at runtime. Note that, such a simple server would not necessarily be operationally compatible with the PM server reference implementation. However, in the static scenario for embedded, the secure communication infrastructure would not be needed since the server would run in the (non-hostile) PHANTOM development environment.

PHANTOM deliverable D2.2 will describe the design and implementation of the NGAC-based system developed for PHANTOM.

4.2.4 Testing results

The current PM reference implementation is running in our lab on a Windows 7 virtual machine. We have source code and are able to modify and rebuild it. We are able to successfully run examples provided with it and to interact with the PM server as and administrator through the PM admin tool and as a user through the PM session manager. The PM currently only runs on Windows and is known to use some Windows-specific libraries. Potential future ports of portions such as the Session Manager and Kernel Simulator to other operating environments will have to deal with these dependencies.

We have a running version of the lightweight ngac policy tool. In our lab it runs on MacOS Sierra under SWI Prolog. As SWI Prolog supports many platforms and there are no OS dependencies in the tool, it is expected that the ngac policy tool is quite portable. The language it supports is a significant part of the policy framework described in the NGAC standard, with the current exception of Prohibitions and Obligations. We have run a number of examples, which are reproduced below.

A graphical representation of two attribute-based policies are shown in Figure 23. 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.

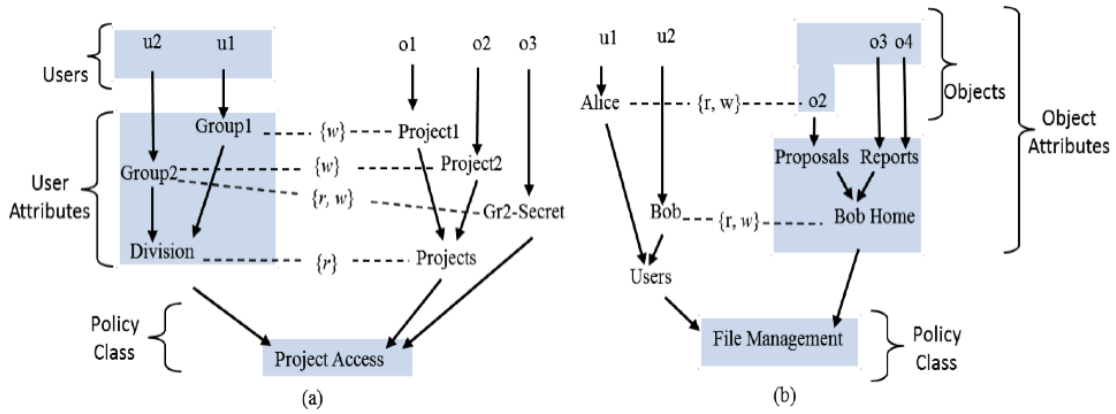


Figure 23: Two attribute-based security policies

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 Figure 24.

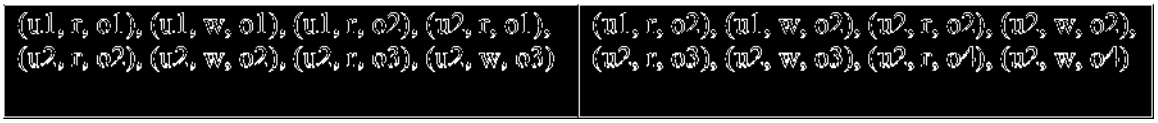


Figure 24: Derived privileges of policies (a) and (b)

A run of the ngac policy tool is shown in Figure 25. 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`.

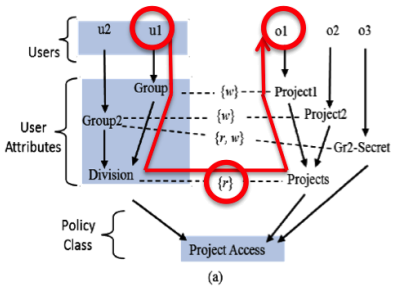
The PM server receives direction from the PM admin tool in the form of imperative commands that build or modify the security policy database. The ngac policy tool can also be able to convert its declarative policy description to the imperative command form used by the PM server reference implementation. 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 25: Description of Policy (a) and interactive test

4.2.5 Dependencies/integration aspects

The PHANTOM application deployment tools, in particular the Deployment Manager, will be expected to generate a policy specification file that will be used at runtime by policy enforcement mechanisms to guarantee that the applications component network architecture is enforced. The developer may optionally provide a supplemental policy in addition to the one automatically generated by the Deployment Manager to further constrain resource access.

PHANTOM applications accessing resources protected by the NGAC DRM will have to be modified to use the NGAC policy enforcement interface rather than directly using the resource access interfaces. PHANTOM components may also use NGAC.

For service oriented or other orchestrated execution and resource access, the service management functions should invoke the policy server to determine that the current policy permits the service combinations it is about to activate.

For each operating environment that is to support NGAC DRM-using processes or DRM-protected resources, a compatible instance of the DRM node manager must be implemented. If the number of operating environments to be supported is excessive or if an environment is particularly difficult to support with a node manager, the DRM-using process may be coded to be the policy enforcing component by having it consult the policy decision component prior to interacting with resources affected by the policy.

This approach could also be used as an interim step prior to availability of a node manager.

4.3 INNOVATIONS BEYOND STATE-OF-THE-ART

The main innovation for Component Network Execution Integrity can be summarized to the seamless integration and orchestration of heterogeneous computing elements while exercising control over the qualities exhibited by their combined operation over a broad range of scale.

For the flexible and uniform Distributed Access Control, the major innovation lies in two axes. First, the Distributed Reference Monitor (DRM) can provide access control over heterogeneous subjects and objects in a highly distributed and heterogeneous computing paradigm. Second, the adaptation of NGAC standard to support lower level access control policy for embedded systems and small target platforms is addressed.

4.3.1 Background technologies utilised in development

The background technologies used are as follows:

- The concepts of MILS including, isolation and information flow control (IIFC), component-based MILS platform, distributed MILS providing IIFC over a collection of networked MILS nodes.
- Next Generation Access Control including, framework for modeling attribute-based access control policies, and a reference implementation of the NGAC standard.

4.3.2 Summary of new technologies/extensions developed

The new technologies/extensions developed are as follows:

- Extended concept of MILS platform to include heterogeneous compute elements such as CPUs, GPUs, and FPGAs and the resource kinds they export.
- Defined the mechanisms to be used in PHANTOM to achieve IIFC for each new compute element type.
- Design and implemented the logic, interfaces, and glue to make the diverse compute element types work seamlessly together as a MILS-style platform and enforce component network execution integrity.
- Defined a declarative language for NGAC attribute-based policies that is easy to derive from policy graphs as shown in the examples.
- Implemented a tool to read the declarative language and answer access control queries based on the policy expressed in the language.
- Investigated the feasibility of extensions as depicted in Figure 22 as "NGAC evolution for PHANTOM", including a lightweight and portable policy server that would accept the declarative language for policy specification, and, optionally also supports the imperative command language of the NGAC reference implementation, thereby enabling dynamically changeable policies, and persistent policy storage.

4.3.3 Early/Full Prototypes functionality

Early prototype of execution integrity

- identify how IIFC achieved for each compute element type
- implement component network realization scheme

Full prototype of execution integrity

- implement/modify mechanisms that provide IIFC for new compute element types (e.g. IIFC for multiple IP cores in an FPGA).

Early prototype of NGAC

- Declarative language for attributed based access control policy specification.
- Lightweight security policy tool to enable development and test of a policy specification and to answer queries concerning the consequences of the policy, the objects visible to a user, the accesses permitted by a user to each visible object, given (user, access_mode, object) is the access permitted under the policy, etc.
- Pre-existing technology did not provide an intuitive declarative machine-processable language in which to express generalized attributed-based access control policies in the NGAC framework.
- Pre-existing technology did not provide a standalone tool with which one could independently develop and test a security policy without deploying it on a live enterprise scale server.

Full prototype of NGAC

- Security policy server with persistent storage of dynamic policy. Can be initialized with a policy developed using the lightweight policy tool. Stores the internalised policy in a persistent store. Can make incremental changes to the policy, which persist in the policy store over restarts.
- Interfaces to permit applications to access resources through policy enforcement points, interfaces for policy enforcement points to query the policy server (combined policy decision point and policy information point) for access control verdicts, and interfaces for policy enforcement point to access protected resources.

5. CONCLUSIONS

5.1 BRIEF SUMMARY

This deliverable presented the first prototypes of the multi-objective mapper, runtime monitoring functionality and security mechanisms providing optimized mapping of application components to heterogeneous platforms.

The technologies described in this deliverable are the ones 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 library: user interface for metrics' configuration, data gathering and metrics storage to the monitoring server, user functionalities to query and analyse the run-time collected metrics
- 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 across enterprise-scale systems that span heterogeneous operating environments.

The above mechanisms collaborate to provide a multidimensional 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.

5.2 CONTRIBUTIONS TO MS3: PRELIMINARY INTEGRATED SYSTEM

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*". According to D5.1 guidelines, the software mechanisms multi-objective mapper, runtime monitoring functionality and security mechanisms described in this deliverable, provide the appropriate APIs and software functions in order to be effectively integrated with PHANTOM use cases. Apart from use case integration, the addressed mechanisms provide the appropriate APIs and aspects for integration between the modules constituting the multi-dimensional optimization framework. The following picture depicts the interacting PHANTOM mechanisms preliminary integration process up to M18.

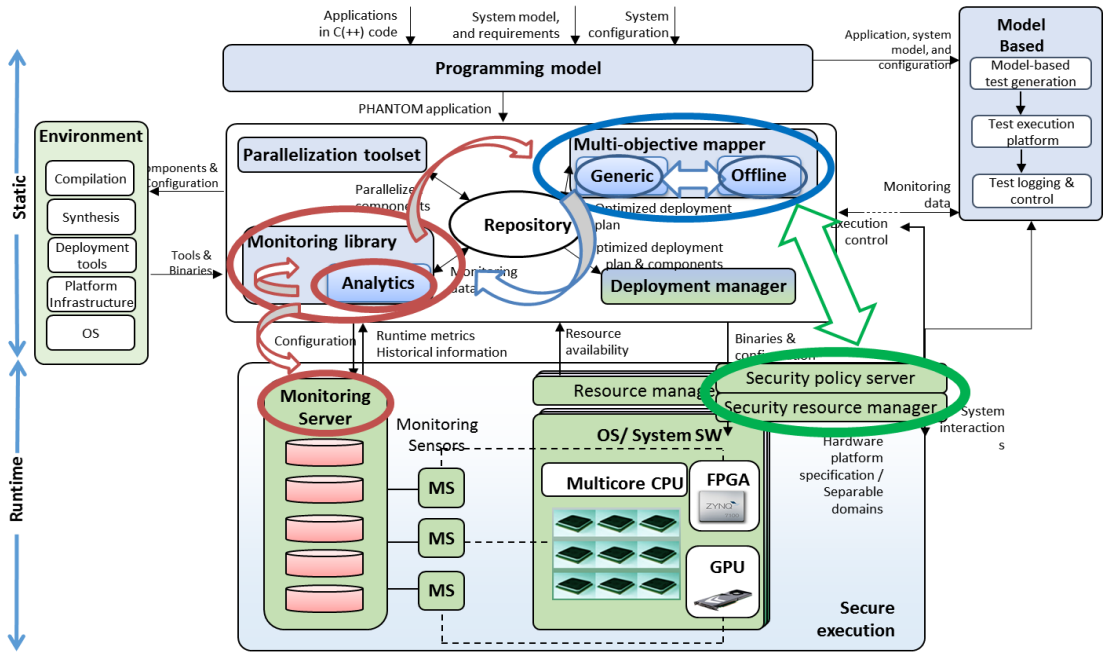


Figure 26: Preliminary integrated system and preliminary versions of HW and SW platforms, up to M18: MS3

The above Figure 26 describes the interaction between the technology components that were described in D2.1. Starting from multi-objective mapper, the first integration will be performed between the high level Generic MOM and the low level Offline MOM (see 5.2.1.1). Then the Generic MOM will also interact with the Monitoring framework in order to acquire the monitoring metrics needed for the optimization process (see 5.2.1.2). Furthermore, MOM will interact with PHANTOM security mechanisms to specify data integration and policy based access (see 5.2.1.3).

5.2.1 First integration activities

5.2.1.1 Multi-Objective Mapper Generic – Multi-Objective Mapper Offline (Only conceptually)

Regarding the integration between Multi-objective Mapper modules (offline and generic), the preliminary integration is performed between the Generic MOM and the offline MOM in conceptual level. Generic MOM provides the offline MOM with the optimized deployment plan and the refined component network in the form of xml documents such as the following figure:

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

    <!--            Component Deployment            -->
    <mapping name="component_B_map" xsi:type="processing" persistency="none">
        <component name="B"/>
        <processor name="P1" cpu-name="CPU1"/>
    </mapping>
```

```
<!--            Communication Objects Deployment            -->
<mapping name="communicationObjectBF1_1_map" xsi:type="communication" persistency="full">
    <component name="output_image_L_1"/>
    <memory name="MEM1"/>
    <comm_protocol name="posix_thread" restriction="user-defined"/><!-- User defined -->
</mapping>
```

Figure 27: Example of optimized deployment plan

From these xml documents the offline MOM will retrieve the selected mapping of application components along with non-functional requirements and achieved performance metrics. Then the offline MOM will perform further analysis and optimization towards timing constraints.

5.2.1.2 Multi-Objective Mapper-Monitoring

Regarding the integration aspects between MOM and the Monitoring framework, the later provides MF client APIs described in Section 3.1 to enable the retrieval of software and hardware metrics per component at runtime. Generic MOM interacts with Repository to retrieve the component network, in order to specify which monitoring requirements are set by the user. As depicted in the following figure, the user adds a corresponding indication in the component network to generate the monitoring in the specific component:

```
<!-- Components Description -->
<component name="A"  type="asynchronous">
    <!--Parallelisation Directives-->
    <PT:parallelisation-directive set-by="PT" name="subcomponents" max_number = "64"/> <!-- Component Parall
    <!--Monitored Computation Time-->
    <MF:monitored-computation-time value = "700" measurement-unit = "ns" />  <!-- Monitored Component Comput

    <!-- Component A Requirements-->
    <requirements set-by="USER" name="component_A_requirements" target="A">
        <!--Non-functional-->
        <non-functional name="CA_WCET" type = "execution-time" max-value = "400" measurement-unit = "ns" />
        <!--Monitoring-->
        <monitoring name="CA_monitor" type = "execution-time" />    <!-- Monitoring Requirement (set by USE
    </requirements>

    <!--C code behavioral description files-->
    <header lang="c" file="CA.h"/>
    <source lang="c" file="CA.c"/>

    <!--Communication Interface -->
    <port name="out" type="out" peer-object="BF1" peer-name="in"/>
</component>
```

Figure 28: Example of monitoring metrics in the refined component network

This indication is required in the XML format as depicted. Regarding the API of the source code, the current implementation of the monitoring client does not support any user defined pragmas but it does offer an API that the programmer can use inside the source code to monitor its preferred metrics (see 3.1.2.1). For sure, the inclusion of pragma in the source code could be also helpful in this case and will be considered in future versions.

The Monitoring framework provides RESTful APIs for sending sampled metrics and timestamps in Json format as described in 3.1.2.4. The Generic MOM can retrieve the sampled metrics through the provided RESTful API in 3.1.4.2 and update the non-

functional metrics values of the application. The following figure provides an example of retrieved monitoring information from MF by MOM:

```
<component name="B" type="asynchronous" >
    <!--Parallelisation Directives-->
    <PT:parallelisation-directive set-by="PT" name="subcomponents" max_number = "1"/> <!-- C
    <!--Monitored Computation Time-->
    <MF:monitored-computation-time value = "700" measurement-unit = "ns" />  <!-- Monitored

    <!--C code behavioral description files-->
    <header lang="c" file="CB.h"/>
    <source lang="c" file="CB.c"/>

    <!--Communication Interface -->
    <port name="in" type="in" peer-object="BF1" peer-name="out"/>
</component>
```

Figure 29: Example of monitoring metrics in the refined component network

As depicted in Figure 29, MOM can provide an HTTP request to the monitoring framework (MF) and update the value of the addressed (monitored) computation time for component with name B".

5.2.1.3 Multi-Objective Mapper/Deployment Manager –Next Generation Access Control

As discussed in section 4.2, interfaces to the policy server would be provided through the PHANTOM Tool Interface (Figure 22), permitting PHANTOM tools such as the MOM or the Deployment Manager to make configuration- or deployment-time queries of the policy. Moreover, the PHANTOM application deployment tools, in particular the Deployment Manager, will be expected to generate a policy specification file that will be used at runtime by policy enforcement mechanisms to guarantee that the applications component network architecture is enforced. The developer may optionally provide a supplemental policy in addition to the one automatically generated by the Deployment Manager to further constrain resource access.

5.2.1.4 Multi-Objective Mapper – Model Based Testing

In order to collect and provide MOM with performance metrics of applications/components over different environments (e.g., CPU and FPGA), a specific MBT-MOM strategy is designed and introduced below. The objective of the specific MBT-MOM strategy 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.

The workflow and implementation status of MBT are detailed in "D3.1 First report on programmer- and productivity-oriented software tools", while the MBT-MOM strategy starts after the creation of MBT models, test generation and test concretization in MBT workflow. The MBT-MOM strategy consists of a list of steps, as follows:

Step 1. MBT firstly gets from PHANTOM Repository two description files, i.e., PlatformDescription.xml, and ComponentNetwork.xml.

Step 2. MBT sends DeploymentPlan.xml and a new ComponentNetwork.xml to the repository. The ComponentNetwork.xml sent by MBT is different from the original file obtained from Repository, the ComponentNetwork.xml sent by MBT contains only description about one component. In this case, PHANTOM will use the new received ComponentNetwork.xml to deploy components to be executed and MBT is able to execute individual component instead of the whole application. Sending DeploymentPlan.xml is necessary for MBT to force the mapping between a component and a certain hardware: in this case, the Deployment Manager takes into account the received DeploymentPlan.xml when deploy tasks to HW platforms.

Step 3. MBT sends input data of applications/components to execute to the Repository.

Step 4. MBT sends the start trigger of execution to Repository, and the platform starts the execution of application/components described in ComponentNetwork.xml following DeploymentPlan.xml.

Step 5. Once the execution is over, the platform sends back the end signal of execution and output data of applications/components to execute.

Step 6. MBT collects the performance metrics regarding each execution from Monitoring Framework and/or PHANTOM repository.

Step 7. MBT sends all collected performance metrics of different executions to the Repository in order to be considered by the MOM.

The general MBT workflow has been implemented as introduced in D3.1, while the development of the MBT-MOM strategy is ongoing along with the integration of PHANTOM Repository, MOM, Deployment Manager and Monitoring Framework.

5.3 NEXT STEPS

The next steps in the multi-dimensional optimization technologies include the complete description of the design and implementation of technologies addressed in WP2 (including the monitoring analytics not addressed in D2.1), along with extensive results and mature integration process that will form the deliverable D2.2-" *Final report on system software for multi-dimensional optimization on heterogeneous systems*" planned for M30.

More precisely:

The Offline MOM will consider exploring more complex architectures and on-chip networks. An additional synthetic use case with strong power constraints based on a hypothetical autonomous vehicle platform will be also used for validation purposes.

The Generic MOM will be refined based on evaluation in PHANTOM use cases. In addition, the next versions of the Generic MOM will consider more non-functional properties including power consumption, memory utilization and others.

The Runtime Monitoring framework will be extended with further plug-ins development and metrics, as well as evaluated in PHANTOM use cases on the target heterogeneous platform, also in terms of performance overhead.

For Isolation of resources and Information Flow Control (IIFC), corresponding strategies will be identified for the heterogeneous processing element types, while it will be elaborated how the strategies of foundational components and operating environment components can be composed, so as to provide coherent global IIFC. For the Next Generation Access Control (NGAC), full determination of supported resource types and operating environments will take place, as well as of the lightweight NGAC server, along with development of the respective resource servers and node manager functionality.

6. 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 Mar. 2015

[2]. DreamCloud project, available online at: http://www.dreamcloud-project.org/, accessed Mar. 2015

[3]. FP7/ICT ALMA project (2011-14): Architecture oriented paraLlelization for high performance embedded Multicore systems using scilAb, available online at: http://cordis.europa.eu/project/rcn/99828_en.html, accessed Mar. 2015

[4]. Davis, Robert I. and Burns, Alan, "A Survey of Hard Real-time Scheduling for Multiprocessor Systems", ACM Computing Surveys, volume 43, number 4, October 2011, http://doi.acm.org/10.1145/1978802.1978814.

[5]. The Eclipse Modelling Framework - http://www.eclipse.org/modeling/emf/

[6]. The Epsilon Project - http://www.eclipse.org/epsilon/

[7]. MAST: Modeling and Analysis Suite for Real-Time Applications - http://mast.unican.es/

[8]. Singh, Amit Kumar, et al. "A Survey and Comparative Study of Hard and Soft Real-time Dynamic Resource Allocation Strategies for Multi/Many-core Systems." *ACM Comput. Surv.* (2017).

[9]. The DreamCloud project - http://www.dreamcloud-project.org/

[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]. Distributed MILS Project - http://www.d-mils.org

[12]. ProSEco Project – http://www.proseco-project.eu

[13]. Wikipedia, "Computer Access Control" (a survey article on access control).

[14]. INCITS 526-2016, "Information technology – Next Generation Access Control – Generic Operations and Data Structures (GOADS)", American National Standard for Information Technology, InterNational Committee for Information Technology Standards Cyber Security technical committee 1, January 2016.

[15]. INCITS CS1/2193-D, "Information technology – Next Generation Access Control – Implementation Requirements, Protocols and API Definitions (NGAC-IRPAD)", American National Standard for Information Technology, InterNational Committee for Information Technology Standards Cyber Security technical committee 1, 29 February 2016.