



ICT Call 4 - ICT for Energy Efficiency
ICT services and software tools enhanced with energy features

FIT4Green

Federated IT for a sustainable environmental impact

Project N° 249020

Energy Control Plug-in for Single Site Data Centres

Software Design Specification

Responsible: Vasiliki Georgiadou (Almende B.V.)

Contributors: Paolo Barone, Robert Basmadjian, Coentin Dupont, Minh Quan Dang, Giovanni Giuliani, Willi Homberg, Ricardo Lent, Juan Carlos Lopez egea, Toktam Mahmoodi, Mikko Majanen, Olli Mämmelä, Thomas Schulze, Andries Stam

Document Reference: D5.1

Dissemination Level: Internal

Version: 1.0

Date: January 24, 2011



EXECUTIVE SUMMARY

The objective of Work Package 5 (WP5) is the development of energy control plug-ins for single- and federated- site data centres. Within the first phase of the project the focus is on single-site data centres and in particular, on designing, implementing and testing energy control plug-ins for single-site data centres within traditional, cloud computing and supercomputing styles.

The goal of the developed plug-in is to dynamically optimize deployment of the applications and services hosted and running at a single-site data centre in order to minimize the energy consumption. The end result of this work package is in fact a prototype that demonstrates the feasibility of the research thesis within the FIT4Green project: “for a data centre with no previous steps with regard to energy optimization, FIT4Green policies and models can provide on average 20% savings in direct server and network devices energy consumption and induce an additional 30% savings due to reduced cooling needs.”

This document provides a single point of reference regarding the design and implementation of the energy control plug-in for single-site data centres. The plug-in prototype software binaries are available as an ISO image for easy installation or live-CD mode deployment. The ISO image is preconfigured so as to demonstrate the use and operation of the FIT4Green plug-in on a simulated cloud environment.

TABLE OF CONTENTS

I. INTRODUCTION	7
I.1. Purpose	8
I.2. Scope	8
I.3. Dependencies and Handovers	8
I.4. Definitions and Acronyms	9
I.5. Outline	9
II. SYSTEM OVERVIEW.....	10
II.1. High Level Architecture	10
II.2. Data Structures	11
II.2.1. Meta-model.....	11
II.2.2. SLA	14
II.2.3. Data centre constraints.....	15
II.2.4. Workload description	15
II.2.5. Deployment action.....	16
II.3. Data and Control Flows	17
II.3.1. Setup	17
II.3.2. Global optimization	17
II.3.3. Resource allocation	17
III. DESIGN CONSIDERATIONS	18
III.1. Assumptions and Dependencies.....	18
III.2. General Constraints	19
III.3. Development Approach	19
III.4. Policies and Tactics	19
IV. COMPONENTS DETAILED DESCRIPTION	21
IV.1. Components Overview	21
IV.2. Core Components	22
IV.2.1. Schemas	22
IV.2.2. Power Calculator.....	23
IV.2.3. Optimizer	28
IV.2.4. Cost Estimator.....	33

IV.3. Plug-in Interface	34
IV.3.1. Manager	34
IV.3.2. Communication (Com-Proxy Pair)	37
IV.3.2.a. Traditional Computing - ENI.....	39
IV.3.2.b. Cloud Computing - HP	40
IV.3.2.c. Supercomputing - JSC	43
IV.3.2.d. Network - ICL	44
IV.4. Web Applications	44
IV.4.1. InitServlet	44
IV.4.2. Web User Interface	45
V. SUMMARY AND OUTLOOK	48
VI. REFERENCES.....	49
APPENDIX A: D5.1 ANNEXES.....	50

List of Figures

Figure 1: FIT4Green work packages and their inter-relations.	7
Figure 2: FIT4Green plug-in as a set of software components that adds energy management capabilities to a data centre.	8
Figure 3: High level architecture illustrating main components along with basic control and data flow.	10
Figure 4: Schematic overview of the FIT4Green plug-in deployment inside a VM.	18
Figure 5: FIT4Green plug-in core components and web applications.	21
Figure 6: The Optimizer's interfaces.	28
Figure 7: Basic data flow of a single workload allocation request.	29
Figure 8: Basic data flow of a global optimization request.	30
Figure 9: Engines inheritance diagram and SLA reader.	31
Figure 10: Algorithms inheritance diagram.	32
Figure 11: Two dimensional linked list.	32
Figure 12: Control flows related to the Monitor and the Controller.	36
Figure 13: ENI Com component and its connection to ENI's servers.	39
Figure 14: Proxy component for the Cloud Computing environment.	42
Figure 15: Com and Proxy component at JSC.	43
Figure 16: Status tab of the FIT4Green Web UI.	45
Figure 17: Configuration tab of the FIT4Green UI.	46
Figure 18: Schematic overview of the FIT4Green UI.	46

List of Tables

Table 1: Document dependencies.....	9
Table 2: Document handovers.....	9
Table 3: SLA parameters.....	15
Table 4: Data centre constraints parameters.....	15
Table 5: Workload description parameters.....	16
Table 6: Deployment action parameters.....	16
Table 7: Interface exposed by the Optimizer.....	29
Table 8: Internal to the Optimizer representation of a server.....	33
Table 9: Internal to the Optimizer representation of a workload.....	33
Table 10: Interface exposed by the Monitor.....	36
Table 11: Interface exposed by the Controller.....	36
Table 12: Interface exposed by a Com component.....	39
Table 13: Operations allowed on a Com component.....	39
Table 14: Annexes accompanying D5.1.....	50

I. INTRODUCTION

The objective of Work Package 5 (WP5) is the development of energy control plug-ins for single- and federated- site data centres. Within the first phase of the project the focus is on single-site data centres. The goal of the developed plug-in is to dynamically optimise deployment of the applications and services hosted and running at a single-site data centre in order to minimize the energy consumption, that is, trying to consolidate load in such a way that some hardware resources can be turned off or at least slowed down.

WP5 aims at creating the interfaces between the energy control plug-in and a broad variety of management systems for data centres. This means that the plug-in is capable of interfacing and interacting with both monitoring and automation frameworks operational at such data centres.

The plug-in is based on energy consumption models and energy optimizing policies, which in turn, are based on energy related information regarding the infrastructure and hardware, including hardware specific energy optimization features. Since these models represent an abstract view of the real situation, eventually they have to be translated into concrete, current representations of specific components within the data centre. In addition, based on monitored energy data of real life situations the models themselves are validated and refined¹. Policies capture knowledge and strategies that provide an active means of improving energy efficiency.

In detail, the plug-in provides access to the existing monitoring systems of the data centre so as to obtain load related data that can then be used to generate realistic energy consumption models of the data centre (WP3). Furthermore, the plug-in allows selecting and enforcing energy optimizing policies for application and service redeployment (WP4). The energy control plug-in is being tested and evaluated within WP6 in specific trial sites for each one of the different computing styles.

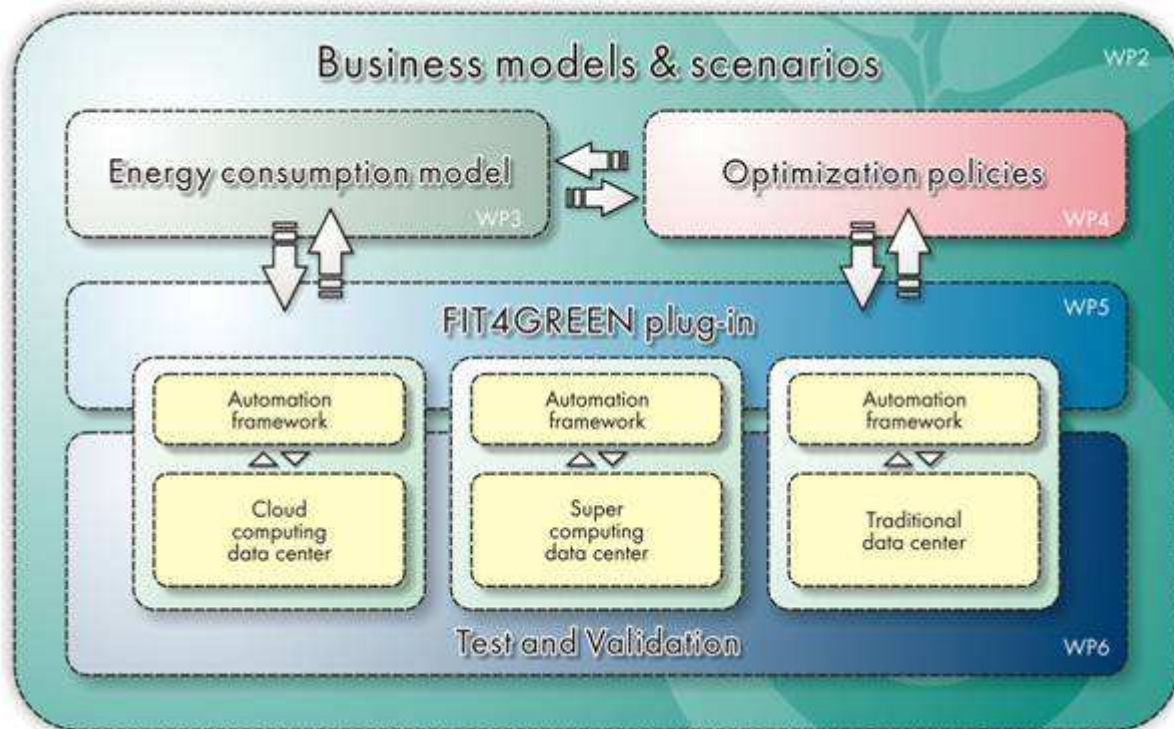


Figure 1: FIT4Green work packages and their inter-relations.

¹ WP6 is to validate the models and report on potential refinements.

I.1. Purpose

The purpose of this document is to provide a single point of reference regarding the design and implementation of the energy control plug-in.

The plug-in prototype software binaries are available as an ISO image for easy installation or live-CD mode deployment.

I.2. Scope

During the first phase of the project the focus has been on designing, implementing and testing energy control plug-ins for single-site data centres within traditional, cloud computing and supercomputing styles. The work is based on the scenarios and use cases identified within WP2.

Figure 2 illustrates the FIT4Green plug-in as a set of software components that adds energy management capabilities to a data centre by interfacing with - *being plugged into* - the management, both controlling and monitoring, and automation system - the so-called *framework* - of the data centre. The latter is managing the data centre resources. The plug-in consists of a set of core components - providing its basic functionality - along with its interface to the data centre framework and the outside world. The idea is to also include an extension manager responsible for easily extending the capabilities of the plug-in itself.

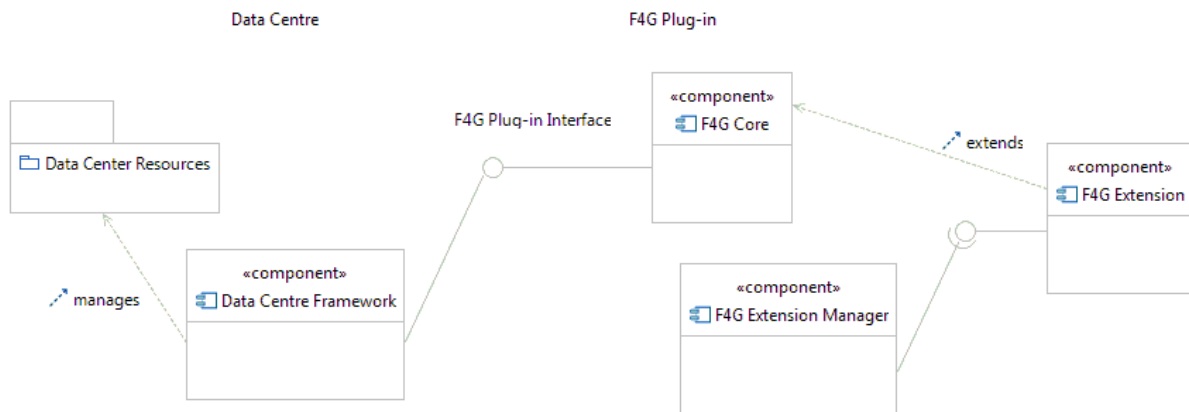


Figure 2: FIT4Green plug-in as a set of software components that adds energy management capabilities to a data centre.

It should be noted that the end result of WP5 is in fact a prototype that demonstrates the feasibility of the research thesis within the FIT4Green project: “for a data centre with no previous steps with regard to energy optimization, FIT4Green policies and models can provide on average 20% savings in direct server and network devices energy consumption and induce an additional 30% savings due to reduced cooling needs.”

Although, during our work we have been following widely used software development best practices², our focus was not on the software process per se, but rather on incorporating the conceptual energy consumption models and optimization policies to a proof-of-concept system that is capable of dynamically managing the applications and services running at a data centre in order to minimise the energy consumption.

I.3. Dependencies and Handovers

Table 1 lists the work packages along with their deliverables that are providing this work with the necessary background information.

² Please refer to Appendix A for a list of related documents.

Work Package	Deliverables	Background information
WP2	D2.1, D2.2	Business requirements, scenarios and use cases; preliminary draft of software requirements; first draft of high level architecture
WP3	D3.1	Energy consumption modelling
WP4	D4.1	Energy optimization policies
WP6	D6.1	Test bed assessment

Table 1: Document dependencies.

Table 2 lists the work packages along with their deliverables that depend on this work.

Work Package	Deliverables	Remarks
WP6	D6.2	Test results and evaluation of 1 st phase
WP2	D2.3, D2.4	Functional requirements and high level architecture

Table 2: Document handovers.

I.4. Definitions and Acronyms

Technical terms and abbreviations commonly used in the FIT4Green project are defined within the FIT4Green Glossary³.

I.5. Outline

The remainder of this document is structured as follows:

Chapter II briefly describes the overall system with references to the high level architecture and its subcomponents, their basic functionality and the main data structures. Connections with the previous work packages, WP2, WP3, and WP4 are made here.

Chapter III outlines:

- the basic assumptions and dependencies regarding the system and its use,
- the global limitations or constraints that have a significant impact on the design of the system along with their associated impact,
- the development approach followed, and
- the design policies and tactics

Chapter IV presents the components detailed design; this includes their interface, class and interaction diagrams, possible algorithmic model and internal data structure, and any other specific to the component information.

Chapter V concludes this document and provides key-point information on future work during phase 2.

At the end of this document you can find a list of annexes that provide details on software development specific topics, such as coding conventions and configuration management plan.

This document is loosely based on the Software Design Specification template proposed by [1].

³ WP5 related updates are included in the FIT4Green Glossary, Version 2.0

II.2. Data Structures

The basic data structures within the FIT4Green plug-in are:

1. Meta-model
2. SLA
3. Data centre constraints
4. Workload description
5. Deployment actions

In the following we present information related to their implementation; for theoretical background please refer to D3.1 and D4.1 deliverables.

II.2.1. Meta-model

The meta-model in FIT4Green is represented in the form of an XML-based document which contains all the energy-related classes and their corresponding attributes. Next, we specify only the necessary attributes of the data structures for the meta-model, where further details can be found in D3.1. It is worthwhile to note we use the quantifier `Array` in order to reflect the fact that the corresponding class contains (has a relationship with) more than one element of the respective array class.

A site is described as follows:

```
SiteType {
  Double PUE;
  Double computedPower;
  Array DatacenterType;
}
```

where `PUE` (Power Usage Effectiveness) is related to the efficiency of the data centre facility, and it's determined as the ratio between the amount of power consumed by the site divided by the power consumed by ICT devices. `computedPower` indicates the power consumption of the site computed through FIT4Green plug-in's power calculator module.

The main parameters of the data centre can be seen in the following:

```
DatacenterType {
  String computingStyle;
  Double computedPower;
  Array RackType;
  Array TowerServerType;
  Array BoxNetworkType;
  Array FrameworkCapabilitiesType;
}
```

where `computingStyle` represents a specific computing style such as Traditional, Supercomputing and Cloud Computing, whereas `computedPower` indicates the power consumption of the data centre computed through FIT4Green plug-in's power calculator module.

A rack is described as follows:

```
RackType {
  Double computedPower;
  Array RackableServerType;
  Array FanType;
  Array RackableNetworkType;
  Array SANType;
  Array EnclosureType;
```

```

    Array PDUType;
}

```

where `computedPower` indicates the power consumption of the rack computed through FIT4Green plug-in's power calculator module.

The following class describes an enclosure:

```

EnclosureType {
    Double computedPower;
    Double measuredPower;
    Array BladeServerType;
    Array NICType;
    Array PSUType;
    Array FANType;
}

```

where `computedPower` and `measuredPower` indicate respectively the power consumption of the enclosure computed through FIT4Green plug-in's power calculator module and the power consumption of the enclosure measured by means of a possible power meter.

A SAN is described as follows:

```

SANType {
    Double computedPower;
    Double measuredPower;
    Array NICType;
    Array PSUType;
    Array RAIDType;
    Array HardDiskType;
}

```

where `computedPower` and `measuredPower` indicate respectively the power consumption of the SAN device computed through FIT4Green plug-in's power calculator module and the power consumption of the SAN device measured by means of a possible power meter.

Power supply unit is described as follows:

```

PSUType {
    Double efficiency;
    Double computedPower;
    Double measuredPower;
    Array FANType;
}

```

where `efficiency` indicates the power efficiency usage which is highly related to the load whereas `computedPower` and `measuredPower` indicate respectively the power consumption of the PSU computed through FIT4Green plug-in's power calculator module and the power consumption of the PSU measured by means of a possible power meter.

A fan is described as follows:

```

FanType {
    Int actualRPM;
    Int width;
    Int depth;
}

```

where `actualRPM` denotes the current rotation speed of the Fan, whereas `width` and `depth` indicate respectively the width and the depth of the Fan.

A server is described as follows:

```

ServerType {
    Double computedPower;
}

```

```

    Double measuredPower;
    Array MainboardType;
}

```

where `computedPower` and `measuredPower` indicate respectively the power consumption of the server computed through FIT4Green plug-in's power calculator module and the power consumption of the server measured by means of a possible power meter. It is worthwhile to note that, `RackableServerType`, `BladeServerType` and `TowerServerType` are specialization of the `ServerType` which represents an abstraction for a generic server computer.

A main board is described below:

```

MainboardType {
    Double computedPower;
    Double memoryUsage;
    Array NICType;
    Array HardwareRAIDType;
    Array CPUType;
    Array RAMStickType;
    Array HardDiskType;
}

```

where `computedPower` indicates the power consumption of the mainboard computed through FIT4Green plug-in's power calculator module, and `memoryUsage` denotes the overall usage of the attached memories whose value is updated constantly through the monitoring module.

The main parameters of the CPU can be seen in the following data structure:

```

CPUType {
    String architecture;
    Double cpuUsage;
    Double computedPower;
    Array CoreType;
    Array CacheType;
}

```

where `architecture` indicates the processor's manufacturer (that is, INTEL, AMD, and so on), `cpuUsage` denotes the utilization of the processor whose value is updated through the monitoring module, whereas `computedPower` represents the power consumption of the processor computed through FIT4Green plug-in's power calculator module.

The main parameters of the core can be seen in the following data structure:

```

CoreType {
    Double voltage;
    Double frequency;
    Double coreLoad;
    Double computedPower;
    Array CacheType;
}

```

where `coreLoad` represents the utilization of the corresponding core whose value is updated through the monitoring module, whereas `computedPower` represents the power consumption of the core computed through FIT4Green plug-in's power calculator module.

The main parameters of the RAM stick can be seen in the following data structure:

```

RAMStickType {
    Int size;
    String type;
    Double frequency;
    Double voltage;
}

```

```

    Boolean isBuffered;
    String vendor;
    Double computedPower;
}

```

where `size` denotes the size of the memory, `voltage` reflects the supply voltage under which the memory operates which are highly dependent on its `type` (that is, DDR1, DDR2, DDR3, and so on), `frequency` denotes the frequency of the memory, `vendor` indicates the manufacturer (that is, KINGSTON, HYNIX, and so on), `isBuffered` shows whether the memory is fully buffered or not. The values for these attributes can be populated inside the data centre model based on the manufacturer's data sheets. Finally, `computedPower` indicates the power consumption of the memory computed through FIT4Green plug-in's power calculator module.

The main parameters of the hard disk can be seen in the following data structure:

```

HardDiskType {
    Double maxReadRate;
    Double maxWriteRate;
    Double readRate;
    Double writeRate;
    Double computedPower;
}

```

where `maxReadRate` and `maxWriteRate` denote respectively the maximum number of read and write operations that can be performed on the disk. The values for both of these attributes can be populated inside the model based on the manufacturer's data sheets. `readRate` and `writeRate` indicate respectively the actual number of read and write operations performed on the disk. The values for both of these attributes are updated constantly through the monitoring module. Finally, `computedPower` represents the power consumption of the hard disk computed through FIT4Green plug-in's power calculator module.

II.2.2. SLA

The main parameters of the SLA can be seen in the following data structure:

```

SLA {
    String ResourceClass;
    Int NumberOfCPUs;
    Double AmountOfMemory;
    Double AmountOfStorage;
    Double BandWidth;
}

```

The parameters `NumberOfCPUs`, `AmountOfMemory`, `AmountOfStorage`, and `BandWidth` are used to describe the specifications of the resource denoted by `ResourceClass` (Table 3).

Name	Type	Meaning
ResourceClass	String	A class of resource in which a specific hardware configuration is predefined; helps simplifying the user's resource selection.
NumberOfCPUs	Int	The number of single core CPUs correlated with the ResourceClass
AmountOfMemory	Double	The amount of RAM correlated with the ResourceClass

AmountOfStorage	Double	The amount of disk space correlated with the ResourceClass
BandWidth	Double	The amount of bandwidth correlated with the ResourceClass

Table 3: SLA parameters.

II.2.3. Data centre constraints

The main parameters of data centre constraints can be seen in the following data structure:

```

Constraint {
    Int MaxvCPUperServer;
    Int MaxvCPUperCore;
    Int MaxVMperServer;
    Int MaxvCPUperVM;
    Double MaxRAMperVM;
}
    
```

These parameters define the resource constraints for servers and virtual machines (Table 4).

Name	Type	Meaning
MaxvCPUperServer	Int	The maximum number of virtual CPU that the server can handle.
MaxvCPUperCore	Int	The maximum number of virtual CPU that a physical core of the server can handle.
MaxVMperServer	Int	The maximum number of virtual machine that the server can handle.
MaxvCPUperVM	Int	The maximum number of virtual CPU that a virtual machine could have.
MaxRAMperVM	Double	The maximum number of RAM that a virtual machine could have.

Table 4: Data centre constraints parameters.

II.2.4. Workload description

The main parameters of workload can be seen in the following data structure:

```

Workload {
    String ResourceClass;
    Int NumberOfCPUs;
    Double AmountOfMemory;
    Double AmountOfStorage;
    Double BandWidth;
    Double PredictCPUUsagerate;
    Double PredictMemUsagerate;
    Double PredictDiskUsagerate;
    Double PredictBandwidthUsagerate;
}
    
```

The parameters NumberOfCPUs, AmountOfMemory, AmountOfStorage, and BandWidth are used to describe the resource requirements of the workload (Table 5). The optimization engine uses these parameters to determine the suitable server to run the given workload. The ResourceClass string is used to point to the SLA data structure which describes the resource requirement. Parameters PredictCPUUsagerate, PredictMemUsagerate,

PredictDiskUsagerate, and PredictBandwidthUsagerate are used to evaluate the power the given workload will consume if it is assigned to a specific server. They help determining the suitable servers that consume least power.

Name	Type	Meaning
ResourceClass	String	The resource class required for the workload; (see II.2.2. for details).
NumberOfCPUs	Int	The number of single core CPUs required for the workload.
AmountOfMemory	Double	The amount of RAM required for the workload.
AmountOfStorage	Double	The amount of disk required for the workload.
BandWidth	Double	The amount of RAM required for the workload.
PredictCPUUsagerate	Double	The predicted CPU usage rate of the workload.
PredictMemUsagerate	Double	The predicted memory usage rate of the workload.
PredictDiskUsagerate	Double	The predicted disk usage rate of the workload.
PredictBandwidthUsagerate	Double	The predicted bandwidth usage rate of the workload.

Table 5: Workload description parameters.

II.2.5. Deployment action

The main parameters of the deployment action can be seen in the following data structure:

```
DeploymentAction {
    String WorkloadId;
    String ActionId;
    String SourceServerId;
    String DestinationServerId;
}
```

Depending on context, some or all of the parameters are used. For example with *Boot* action, only *ActionId* and *DestinationServerId* are used. However, with *MoveVM* action, all parameters are used (Table 6).

Name	Type	Meaning
WorkloadId	String	The workload identification.
ActionId	String	The action identification.
SourceServerId	String	The server identification that is running the workload.
DestinationServerId	String	The server identification that will be affected by the action.

Table 6: Deployment action parameters.

II.3. Data and Control Flows

There are 3 basic data and control flows:

1. Setup
2. Global optimization
3. Resource allocation

II.3.1. Setup

The FIT4Green user, the data centre operator, setups and configures the plug-in. This is performed once during first time launch and is valid until data centre infrastructure changes. The setup includes three steps:

Step 1: Edit/Import static model information and create serialized model instance

Step 2: Edit (technical level) SLAs, software deployment constraints and rules

Step 3: Select optimization policies and supply configuration information if needed

These steps can be accomplished either via the FIT4Green web UI or directly editing the relevant configuration files via a simple text editor.

II.3.2. Global optimization

Global optimization is performed in a continuous loop:

Step 1: The monitoring component of the plug-in updates the dynamic attributes of the current model instance with information acquired from the data centre monitoring tools.

Step 2: The plug-in evaluates when it is worth running the optimizer and triggers a global optimization request

Step 3: The optimizer suggests a set of actions to change to a better system state from energy perspective

Step 4: The plug-in sends actions list to the data centre automation framework

Step 5: Go to *step 1* and continue this loop

II.3.3. Resource allocation

Resource allocation is triggered only on request - by the data centre automation and management framework:

Step 1: The automation framework needs to decide where to allocate a new workload

Step 2: The plug-in forwards the request to the optimizer

Step 3: The optimizer responds with an “optimal resource allocation”

Step 4: The plug-in communicates the response to the framework, which will implement the resource allocation

III. DESIGN CONSIDERATIONS

III.1. Assumptions and Dependencies

The end user of the plug-in is the data centre operator. The end user should be able to interact with the plug-in via a Graphical User Interface (GUI), which also guides the editing of simple ASCII files - either in text or XML format.

Following the web application paradigm, it has been decided to provide a web-based UI; the latter is developed based on Asynchronous JavaScript and XML (AJAX) technology and in particular Google Web Toolkit (GWT). For details on the FIT4Green UI see section IV.4.2.

GWT requires a web server with servlet support; to this end it has been decided to use Apache Tomcat⁴, which can also manage web services with REST (Representational State Transfer) or SOAP (Simple Object Access Protocol) - through Apache Axis.

The plug-in itself should be easy to deploy and energy efficient at runtime; to this end the idea is to create a Virtual Machine (VM) which contains all the integrated components: a VM Appliance, a sort of “black box” that you simply turn on to get a job done, without spending much time in installing software.

Figure 4 illustrates how the FIT4Green plug-in is to be deployed inside a VM. The VM - created either by using VMWare or VirtualBox technology⁵ - runs Linux Ubuntu operational system. Once the VM is started, the built-in Tomcat server is launched pre-configured so as to run the FIT4Green plug-in. The FIT4Green web UI provides the plug-in’s user with the necessary interface to overview and manage the plug-in’s operation.

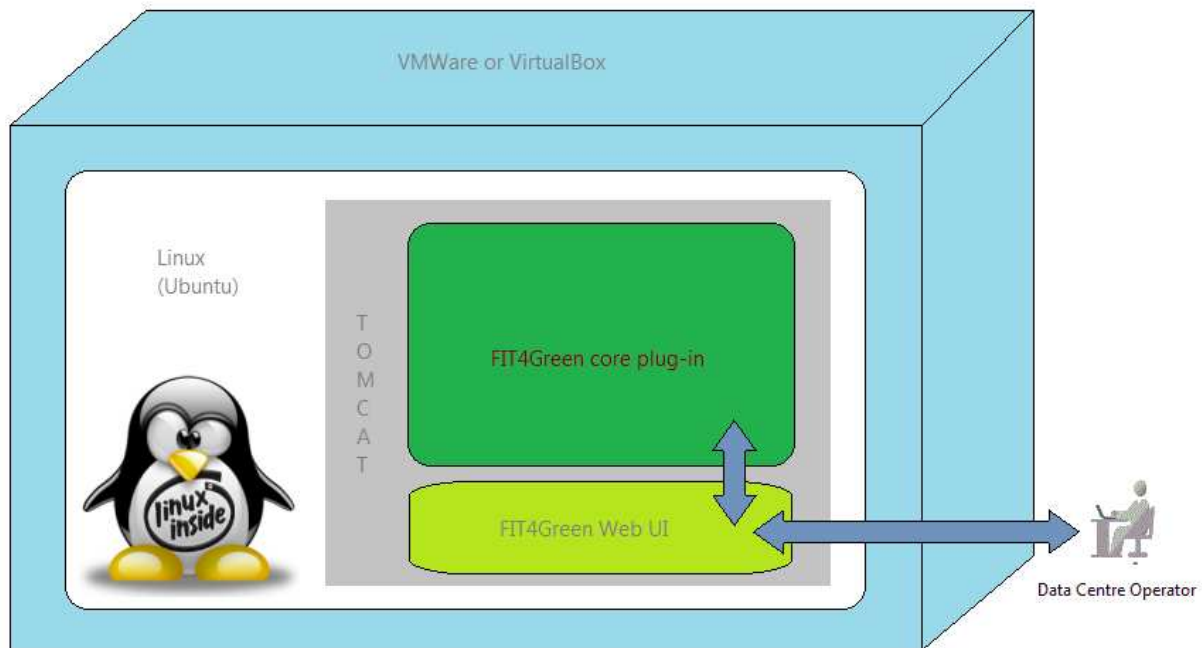


Figure 4: Schematic overview of the FIT4Green plug-in deployment inside a VM.

For the time being though, and also due to license issues regarding which VM technology to be used, we package the first release of the prototype (V1.0) for WP6 use as an Ubuntu

⁴ At time of development, latest stable version of Apache Tomcat was 6.0. For best results we are using JRE 6.

⁵ The relevant decision at consortium level is still pending.

based Linux ISO image. Each test-bed prepares a machine - either physical or a VM^{6, 7} - capable of running Ubuntu and boots from the ISO image to install FIT4Green. The ISO image is preconfigured so as to demonstrate the use and operation of the plug-in on a simulated cloud environment.

For detailed information on architectural guidelines and requirements see D2.1.

III.2. General Constraints

At consortium level it has been decided that all software development tools to be used with FIT4Green plug-in should be open source licensed - without contaminating the FIT4Green source code (for example no General Public License). It should be noted that for the plug-in itself open source disclosure is only for selected components and not for the full solution.

For information regarding the programming languages as well as the simulation and modelling tools used within FIT4Green, please refer to D5.1 - Annex 4: Languages and Tools.

III.3. Development Approach

The development approach followed within FIT4Green is based on an iterative and evolutionary development process, adapted to fit the project's needs and constraints, such as short runway and geographically distributed team.

Each WP5 development stage is split into a series of variable-time iterations. At the beginning of an iteration, a simplified version of the main use cases presented within D2.1 is extracted. The WP5 group works towards developing - that is, designing, coding, building and testing - a "stand-alone" system that accomplishes the tasks described in the selected simplified use case. At the end of the iteration there is a system ready for evaluation.

Next iteration selects other aspects of the main use cases and evolves the end-product of the previous iteration. In the beginning of the first phase, the iterations continued for several weeks; as the development was moving forward the duration of iterations was minimised to 1 week.

The main benefit of following this process is that we have early on a tangible software product to work with; this helps us to anticipate bottlenecks and spot down misconceptions and risks.

FIT4Green software development is based on Eclipse⁸ Integrated Development Environment (IDE). For detailed information regarding the development environment along with instructions on how to configure it and information on the various projects, please refer to D5.1 - Annex 3 Project Configuration Description.

III.4. Policies and Tactics

We have followed the so-called GRASP⁹ guidelines, which are all about assigning responsibilities to classes and objects in object-oriented design. All components are therefore assigned to clear roles and thus the number of interactions between them is minimized.

⁶ The VM is created using a site-preferred hypervisor.

⁷ The preferred approach is the one based on the VM: the energy impact of the plug-in is lower when running on a VM than when running on a physical machine.

⁸ Both Helios or Galileo

⁹ Acronym stands for General Responsibility Assignment Software Principles.

In addition, we have based our designs on the following simple policies:

- Build an effective support for FIT4Green developers
- Apply “encapsulation” principle when possible
- Apply “isolation” principle when possible
- Keep in mind future “contributions “ and “extensions”
- Consider Web Service integration
- Do not over-engineer

In order to ensure the readability and maintainability of the software, allowing engineers to understand existing code, spot potential bugs and add new features more quickly and thoroughly, we have decided to adjust well known coding conventions to our project’s needs and requirements. Please refer to D5.1 - Annex 1: Coding Conventions for more information.

Subversion (SVN) is chosen as the version control system. D5.1 - Annex 2 presents the FIT4Green configuration management policies.

IV. COMPONENTS DETAILED DESCRIPTION

IV.1. Components Overview

Figure 5 shows the FIT4Green plug-in’s core components and web applications along with basic control and data flow

Init Servlet is an HTTP Servlet which launches the FIT4Green plug-in upon Tomcat start-up. Main is the entry point to the plug-in which provides all the interfaces exposed by the other components. This module can be thought of as a directory or a factory that, upon plug-in’s initialization, starts up the threads of all its core components. It is also responsible for stopping the plug-in upon request.

The usual plug-in operation follows a “sense-interpret-act” loop. The monitor “senses” any changes from the framework - either data to update the model or requests to send to the engine. The optimizer “interprets” the current state of the data centre based on the energy consumption models integrated into the power calculator and decides on the optimal deployment strategy. The list of suggested actions is forwarded to the controller which in turn propagates them to the corresponding com. The latter implements the low level communication with the data centre automation and monitoring tools, closing the loop when it “acts” upon the data centre resources.

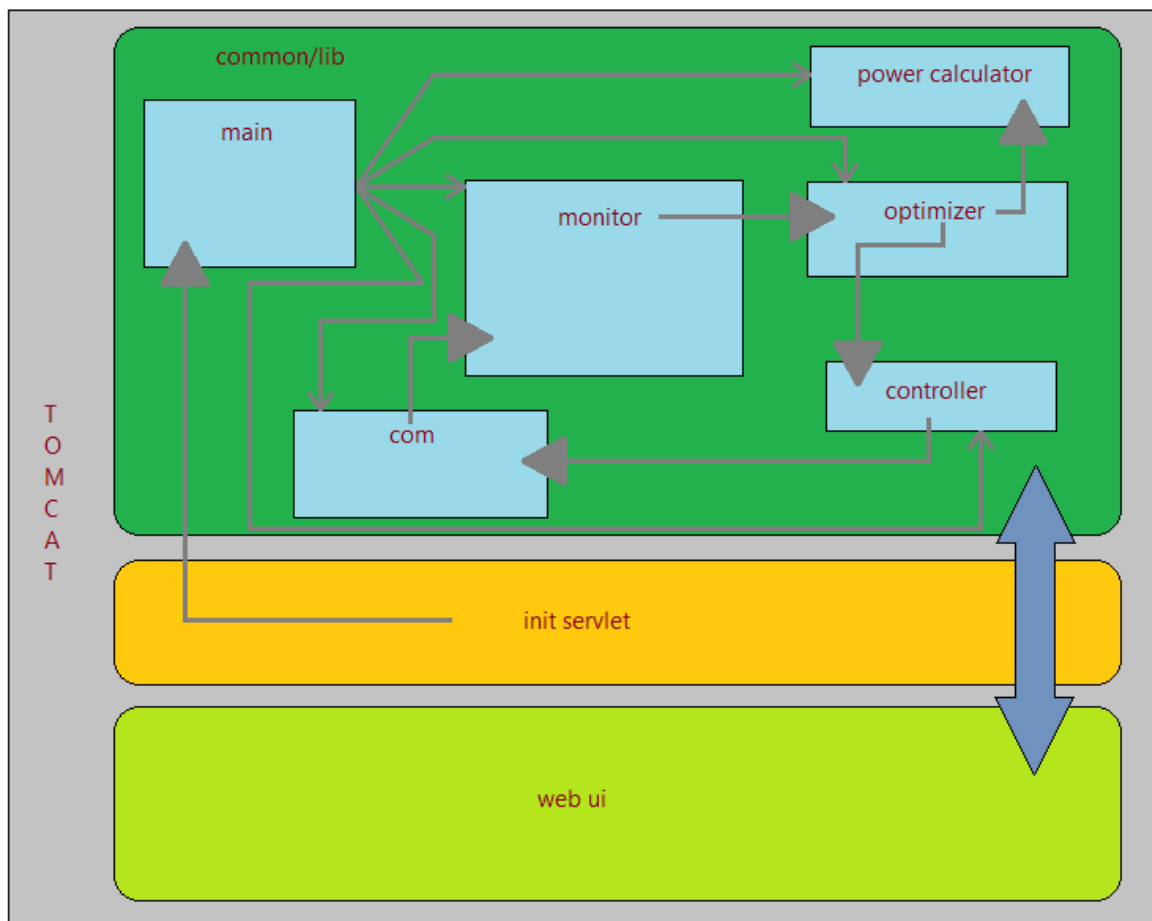


Figure 5: FIT4Green plug-in core components and web applications.

In the following sections we are discussing in detail the various FIT4Green components.

IV.2. Core Components

IV.2.1. Schemas

All the persistent data structures (see also section II.2.) and the macro interfaces used in FIT4Green are XML-based and comply with well-defined XSD schemas.

At application level, specific technologies have been selected to enable the manipulation of the data represented in XML.

To facilitate the serialization/de-serialization and access to XML data we adopted the JAXB technology, which provides tools and libraries for automatically transforming XML-schemas into Java class hierarchies and for manipulating them. After applying such transformation to the FIT4Green XML schemas, the FIT4Green data model and the macro interfaces representation are made available to the software developers in form of Java classes.

The JAXB libraries provide also facilities for automatically creating Java objects at runtime from XML files compliant with the above mentioned schemas. This allows developers to handle the XML data as Java objects at runtime, which is more convenient for software developers and also faster on the computational side. Obviously, the JAXB technology provides means to transform the java objects back into XML files for persistent storage when needed.

Once the java objects representing XML data are created at runtime, it is typically required to look for specific field contents based on search conditions. For this purpose, we adopted the XPath technology, which allows applying the standard XPath query language directly to Java objects. The power of the XPath technology for the developer relies in the possibility of leveraging complex features of the XPath language (such as the queries) without having to handle directly specific code for parsing XML. In addition, it allows modifying the contents of object fields by simply specifying XPath expressions.

The FIT4Green meta-model has been defined during the WP3 activities. To this end, WP3 designed a set of UML class diagrams representing the FIT4Green meta-model. Afterwards, by leveraging specific tools, such class diagrams have been converted into XML schemas. Due to limitations in the capabilities of the translation tools available, the XML schemas generated required some manual corrections. Finally, during the development phases, the schemas have been further refined, to enable the FIT4Green plug-in algorithms to navigate the data object structures.

The FIT4Green model, stored as an XML file, is loaded and de-serialized (transformed into a Java object representation) by the Monitor component at start up time. Therefore, the object representation is dynamically modified at runtime in a consistent way by the Monitor on behalf of the active communication components (each one representing a computing style) in the plug-in.

The schemas for SLAs, Constraints and Workload Description, and Deployment Actions have been entirely created in WP4 and directly imported in the development environment. Their corresponding configuration file are loaded and de-serialized by the Optimizer component.

All the FIT4Green XML-schemas are grouped into a self-contained Java project. This project automatically generates the corresponding java packages/classes and collects them into a jar file at build time.

The jar file is then included by the classpath of all the other Java projects containing the source code for a FIT4Green component. This means that the source code of the automatically generated java classes is never modified by a developer. Every modification can only be made at XML-schema level.

IV.2.2. Power Calculator

The Power Calculator module contains specific power consumption prediction models and formulas based on the state of the data centre. The theoretical background on these models and formulas is provided by the FIT4Green meta-model, which is represented in the form of a tree, and has the central role of describing all ICT resources (both physical and virtual machines as well as software applications) inside the data centre, their interconnections and the respective load applied on the infrastructure. In the following, we first provide the interfaces with which the power calculator module can be referenced, and then outline its algorithmic model.

The interface

The power calculator is a stateless module providing the following main functions as public interfaces used by the Optimizer module:

- `computePowerKeyword(KeywordType param): PowerData`

This is a generalization of all types of functions (ended with *Keyword*) that take in as a parameter `param` of type *Keyword* and returns a data structure that contains the power consumption of the whole *Keyword* system in Watts. The followings are the possible values of the quantifier *Keyword*:

- FIT4Green
- Site
- Datacenter
- Rack
- Server
- RAID
- HardDisk
- SolidStateDisk
- Fan

- `computePowerCPU(CPUType cpu, OperatingSystemType operatingSystem): PowerData`

The above function takes in as parameters the necessary information related to a processor as well as the running operating system and returns a data structure that contains the power consumption of the whole processor in Watts.

- `computePowerCore(CoreType core, int numberOfCores, CPUArchitectureType cpuBrand, OperatingSystemTypeType operatingSystem): PowerData`

The above function takes in as parameters the necessary information related to a core as well as the running operating system and returns a data structure that contains the power consumption (in Watts) of the individual core of a processor.

- `computePowerMainboardRAMs(MainboardType mainboard): PowerData`

The above function takes in as a parameter the necessary information related to the installed memory modules and returns a data structure that contains the power consumption (in Watts) of the total installed memory.

Algorithmic model

Next, we present a short description (pseudo-code) of the general algorithm of the public and private function of the power calculator.

- **computePowerFIT4Green**(*FIT4GreenType model*):
 1. Traverse the tree model starting from the root
 - a. While the tree contains nodes of type *Site*:
 - i. Call **computePowerSite**(*SiteType*)
 - ii. Add the result of (i) to the overall computed power consumption
- **computePowerKeyword1**(*Keyword1Type param*):
 1. Traverse the tree starting from the node of type *Keyword1* (taking the values *Site*, *Datacenter* and *Rack*)
 - a. While the tree contains nodes of type *Keyword2* (taking the values of *Datacenter* for *SiteType*, *NetworkNode*, *Rack* and *Server* for *DatacenterType*, and *PDU*, *Enclosure*, *SAN*, *NetworkNode*, *Fan* and *Server* for *RackType*):
 - i. Call **computePowerKeyword**(*KeywordType*)
 - ii. Add the result of (i) to the overall computed power consumption
 2. Assign the overall computed power to "computedPower" of param
- **computePowerEnclosure**(*EnclosureType enclosure*):
 1. Traverse the tree starting from the node of type *Enclosure*
 - a. While the *Enclosure* contains nodes of type *PSU*:
 - i. Count number of PSUs and test whether each PSU has a load greater than zero
 2. If all *PSUs* have a load of zero
 - a. Set the overall computed power to zero
 3. If a *PSU* has a load greater than zero
 - a. While the *Enclosure* contains nodes of type *Keyword* (taking values of *NIC*, *Fan*, *Server*):
 - i. Call **computePowerKeyword**(*KeywordType*)
 - ii. Assign the result to "computedPower" of *Keyword*
 - iii. Add the result of (i) to the overall computed power consumption¹⁰
 - b. While the *Enclosure* contains nodes of type *PSU*:

¹⁰ For space considerations, in the coming functions we only represent the formulas instead of repeating steps i, ii, and iii

- i. If PSU's "measuredPower" is given
 - Assign the measured value to "computedPower" of PSU
 - Add the result to the overall computed power consumption
 - ii. If Enclosure's "measuredPower" is given
 - Distribute evenly the measured power of the enclosure among the PSUs
 - Compute the power of PSU based on this measured power and its efficiency
 - Assign the result to "computedPower" of PSU
 - Add the result to the overall computed power consumption
 - iii. If no information
 - Distribute evenly the computed power of the enclosure among the PSUs
 - Compute the power of PSU based on this computed power and its efficiency
 - Assign the result to "computedPower" of PSU
 - Add the result to the overall computed power consumption
 - 4. Assign the overall computed power to "computedPower" of enclosure
- **computePowerServer(ServerType server):**
 1. Traverse the tree starting from the node of type *Server*
 - a. While the *Server* contains nodes of type *PSU*:
 - i. Count number of PSUs and test whether each PSU has a load greater than zero
 2. If all *PSUs* have a load of zero and server is not blade
 - a. Set the overall computed power to zero
 3. If a *PSU* has a load greater than zero
 - a. Fetch the operating system
 - b. While the *Server* contains nodes of type *Mainboard*:
 - i. **computePowerMainboard(MainboardType, OperatingSystemType)**
 - c. If *Server* is of *TowerServerType* or *RackableServerType*

- iv. Add the result to the overall computed power consumption of CPU
 - b. If CPU has 4 cores and of type AMD
 - i. Add a factor to the overall computed power consumption of CPU
 - c. Else
 - i. Reduce the overall computed power consumption of CPU by some factor
- 2. Add the idle power of CPU to the overall computed power consumption of CPU
- 3. Assign the overall computed power to "computedPower" of CPU
- **computePowerMainboardRAMs(MainboardType):**
 - 1. Traverse the tree starting from the node of type *Mainboard*
 - a. While the *Mainboard* contains nodes of type *RAMStick*:
 - i. Add the size of the *RAMStick* to the total size of installed RAMs
 - ii. Count number of RAMs
 - b. If the processor is in idle state
 - i. Assign a value of zero to the probability of accessing the memory
 - c. If the processor is not in the idle state
 - i. Divide the size of used RAMs by the total size of installed RAMs
 - ii. Assign the result of (i) to the probability of accessing the RAMs
 - iii. From the result of ii:
 - a. Give higher probability to the part of the memory occupied by the OS by dividing the answer of ii into half
 - b. Distribute evenly the other half among the other RAMs not occupied by the OS.
 - d. While the *Mainboard* contains nodes of type *RAMStick*:
 - i. If the OS occupies the *RAMStick*
 - a. Compute the power consumption of the memory using the accessing probability of iii.a
 - ii. Else
 - a. Compute the power consumption of the memory using the accessing probability of iii.b

- iii. Assign the power consumption of the RAMStick to the "computedPower" of RAMStick
 - iv. Add the power consumption of the RAMStick to the overall computed power consumption of memories
- **computePowerHardDisk(HardDiskType):**
 1. Traverse the tree starting from the node of type *HardDisk*
 - a. If there are neither read nor write operations
 - i. Compute the idle power of the disk
 - b. Else
 - i. Compute the disk's accessing probability by dividing the current number of read and write operation by the maximum number of read and write operations
 - ii. Compute the power of the hard disk by taking into account its accessing, idle and startup states based on (i)
 2. Assign the result to the "computedPower" of the HardDisk

Based on the FIT4Green model instance with its constantly updated values, the power calculator is capable of computing the power consumption of the ICT resources of data centres. It is worthwhile to note that the accuracy of power consumption prediction is based on the level of information that the monitoring module provides to the power calculator module.

IV.2.3. Optimizer

The optimization engine is a stateless module. It interfaces with other components as shown in Figure 6.

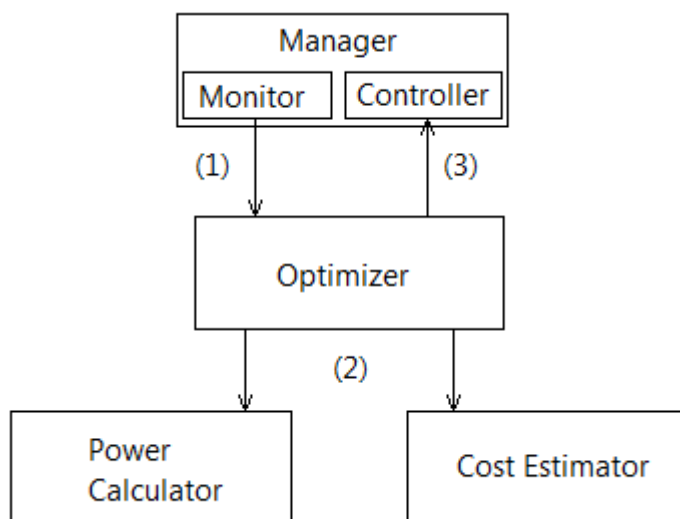


Figure 6: The Optimizer's interfaces.

The Manager, through the Monitor, calls the Optimizer to respond to the optimization requests. The Optimizer calls the Power Calculator and the Cost Estimator during the

optimization process to acquire information on power consumption of data centre resources. Based on this information the Optimizer suggests a list of actions that can potentially lead to reduced energy consumption. The list of the suggested actions is sent back to the Manager, and in particular to the Controller - which propagates it further to the corresponding Com component. For details regarding the Manager (both Monitor and Controller), the Power Calculator and the Cost Estimator see IV.3.1. , IV.2.2. , and IV.2.4. respectively.

The optimizer exposes two main functions as interfaces to the Monitor which are shown in Table 7.

allocateResource(Fit4GreenAllocationRequestType, Fit4GreenType, Fit4GreenType, Fit4GreenType): Fit4GreenAllocationResponseType
<p>Handles a request for resource allocation</p> <p>Parameters:</p> <p>allocationRequest the data structure describing the workload</p> <p>model the data structure describing the FIT4Green model</p> <p>SLA the data structure describing associated SLA</p> <p>constraint the data structure describing the associated data centre constraints</p> <p>Returns:</p> <p>A data structure representing the result of the allocation, that is, the list of suggested actions</p>
performGlobalOptimization(Fit4GreenType, Fit4GreenType): boolean
<p>Handles a request for a global optimization</p> <p>Parameters:</p> <p>model the data structure describing the FIT4Green model</p> <p>constraint the data structure describing the associated data centre constraints</p> <p>Returns:</p> <p>true if the global optimization starts successfully, false otherwise; the result of the global optimization is transferred to the Controller inside the function</p>

Table 7: Interface exposed by the Optimizer.

Single workload allocation request

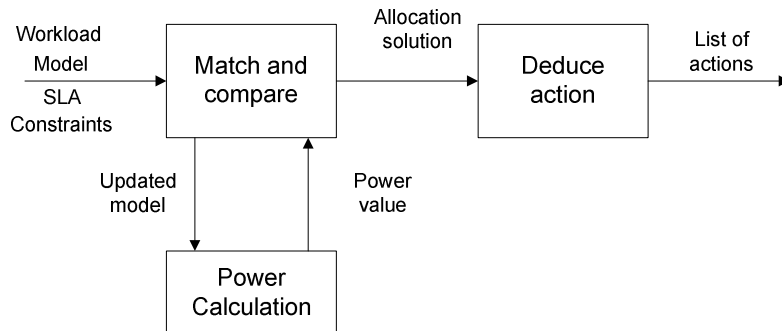


Figure 7: Basic data flow of a single workload allocation request.

The input data for a single allocation request includes the workload description, model description, SLA description and data centre constraints description (see also section II.2.). Based on this information, the Optimizer finds suitable candidates to run the workload by

matching resource requirements to appropriate resource description. For each candidate, the Optimizer assumes to assign the workload to that candidate, update the model and calculate the expected power consumption. The candidate to be selected should have the minimum expected power consumption. Further details regarding the algorithm can be found within D4.1. After finding out the appropriate candidate, the Optimizer generates a list of actions to be sent to the Controller.

Global optimization request

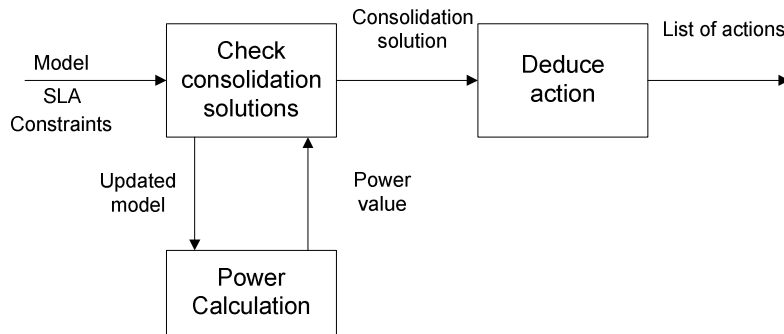


Figure 8: Basic data flow of a global optimization request.

The input data for the global optimization request includes the model description, SLA description and data centre constraints description (see also section II.2.). Based on this information, the Optimizer looks for a variety of consolidation solutions. For each consolidation solution, the optimizer updates the model and calculates the expected power consumption. The suitable solution should have the minimum expected power consumption. Further details regarding the algorithm can be found within D4.1. After finding out the appropriate consolidation solution, the Optimizer generates a list of actions to be sent to the Controller.

Engine choice

Once called, the optimizer calls the corresponding engine, according to the type of the data centre: “Cloud”, “Tradition¹¹”, and “HighPerformanceComputing¹²”. For each data centre type, the Optimizer calls dedicated optimization algorithms. The working mechanism of each optimization algorithm can be found within D4.1

Engine structure

The optimizer engine is split into:

- The engine for HPC,
- The engine for Traditional,
- The engine for Cloud.

These engines are responsible for marshalling and un-marshalling the input data as well as their response. Each of them calls the respective optimization algorithms.

The Cloud engine utilizes a class called *SLAReader* to read the SLA. The SLA is expressed in the form of an XML-based file. Based on the corresponding schema definition, it is possible to read and de-serialise this file - using JAXB tool (see also IV.2.1.).

¹¹ Stands for traditional.

¹² Alternate name to supercomputing.

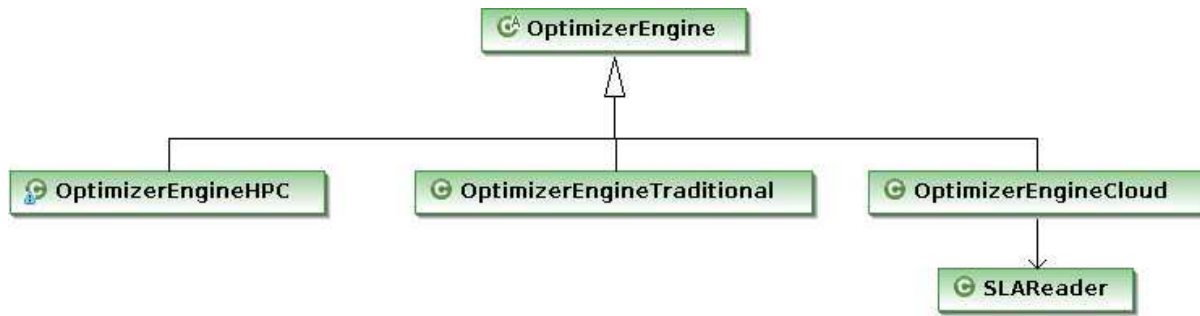


Figure 9: Engines inheritance diagram and SLA reader.

The inheritance diagram shown in Figure 9 has been designed so as to be able to put every common code among the three engines into the parent class, `OptimizerEngine`. Only specific code is located in the child-classes `OptimizerEngineHPC`, `OptimizerEngineTraditional`, and `OptimizerEngineCloud`.

In the following we provide details on specific implementation issues related to each optimization engine type; the cloud and traditional computing cases are handled together due to similarities they present.

High Performance Computing Optimization Engine

The High Performance Computing (HPC) optimization engine is used for scheduling jobs that are received by the Resource Management System (RMS) of the data centre environment. Scheduling can be carried out by three scheduling algorithms: priority FIFO, backfill first fit and backfill best fit. The specific scheduling algorithm is selected by the data centre operator. The primary goal of the optimization engine is to schedule jobs appropriately according to the selected scheduling algorithm, and reduce energy by shutting down resources that are in an idle state when not needed for job execution.

The optimization engine is called every time something changes in the data centre, for example a job is started or some nodes are shut down or hibernated. Inside `runGlobalOptimization(model)` function, the optimizer takes the FIT4Green model as a parameter, and reads all the necessary information from it. This information is gathered to three lists: one for servers (`List<ServerType> srvList`), one for queued jobs (`List<JobType> jobList`), and one for jobs that are currently running (`List<JobType> runningList`). Next the `jobList` is traversed and checked whether some jobs can be started on some nodes, or perhaps some servers can be set to sleep/power off mode. Finally, actions, such as start job, power on/off node, are created and sent to the Controller. Jobs are started only on idle servers, not sleeping/power off servers, so as to start a job on sleeping/power off node(s), the node is first waken up by sending power on action. After that there should be a new `runGlobalOptimization()` call (now with enough idle nodes for starting the job).

Cloud and Traditional Computing Optimization Engine

The algorithms themselves are structured as shown the inheritance diagram of Figure 10.

This structure has been chosen to allow maximal reuse of common parts. Indeed, the algorithms between Traditional and Cloud share several common features.

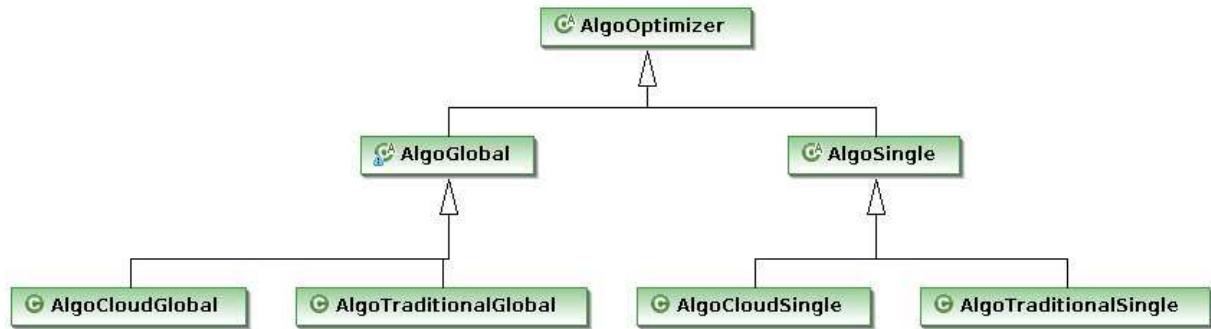


Figure 10: Algorithms inheritance diagram.

To perform the optimization, two dimensional linked lists are used as presented in Figure 11.

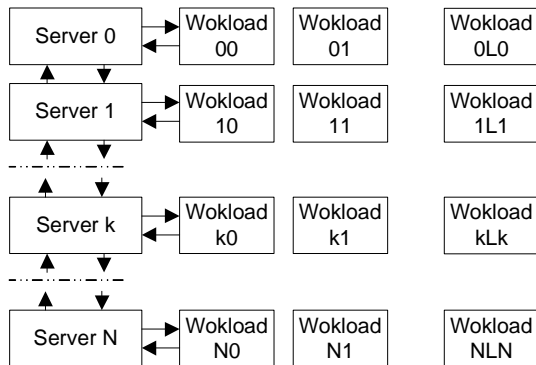


Figure 11: Two dimensional linked list.

This data structure is used to describe the load distribution of the data centre. The data centre has an array of server indexed from 0 to N. With each server index i, there is an array of workloads indexed from 0 to Li. During the optimization, we change the workload distribution to find the most energy efficient solution.

The optimizer uses an internal representation of a server as shown in Table 8.

```

public class OptimizerServer extends ServerType implements Cloneable {
    ...
    /** State of the optimizer during optimization:
     * FREE = empty server ready for switch off
     * SOURCE = the optimizer will try to empty this server
     * TARGET = the optimizer will use this server to put the VMs
     * NOT_ASSIGNED = no role is yet decided.
     */
    public static enum CandidateState { FREE, SOURCE, TARGET,
        NOT_ASSIGNED};

    CandidateState candidateState;
    /** Working list of workloads, used during optimization.
     */
    protected List<OptimizerWorkload> workloads;
    public List<OptimizerWorkload> getWorkloads();
    public void setWorkloads(List<OptimizerWorkload> workloads);
    ...
    /** Server constructor for Cloud
     */
}
    
```

```

public OptimizerServer(ServerType modelServer, VMType myVMTypes)
    throws CreationImpossible;
/** Server constructor for traditional
 */
public OptimizerServer(ServerType modelServer)
    throws CreationImpossible;
...
}

```

Table 8: Internal to the Optimizer representation of a server.

The optimizer also uses its own representation of a workload, as show in Table 9.

```

public class OptimizerWorkload extends VirtualMachineType {
...
/** Workload from a VirtualMachineType from model (Traditional)
 */
public OptimizerWorkload(VirtualMachineType VM)
    throws CreationImpossible;
/** Workload from a allocation type (Traditional)
 */
public OptimizerWorkload(TraditionalVmAllocationType allocation,
    String frameWorkID) throws CreationImpossible;
/** Workload from a VM from the SLA (Cloud)
 */
public OptimizerWorkload(VMAttrType VM, String frameWorkID);
...
}

```

Table 9: Internal to the Optimizer representation of a workload.

Limitations and constraints

We plan introducing strong typing for physical measures so as to enforce dimension checking during compilation, and thus avoid dimension/unit mismatch between components and internal components.

Language describing data centre constraints, SLA, is to be extended so as to apply to other contexts; we plan updating it in order to reflect the specific aspect of that context.

Optimization algorithms are to be extended so as to apply to other contexts; we may revisit the algorithm according to the new context.

Optimization engine is to be further tuned so as the integration can move smoothly.

IV.2.4. Cost Estimator

A network's power consumption can be expressed as a function of the data traffic handled by the system. In particular, this power consumption will be determined by the offered load at each network device (router, switch, and so on). The load includes both the packet inflows and outflows at the device. To compute the network power consumption, it is also necessary to have access to a description of the network topology, which indicate the way network devices and servers are connected. Knowledge of the link line rates and nodes' processing bandwidth are also needed.

We have considered 2 types of operations for the network cost estimation. The first type allows estimating a whole network power consumption for a given traffic load. For this, the

input is a set of packet flow descriptions, corresponding to the traffic expected to be present in the network. Each packet flow description gives information about the source and destination of the flow, and the flow bandwidth (in packets per second). A flow description represents either an individual communication or a communication aggregate. These communications could occur either between two servers within the local area network or between one internal server and the Internet. In the latter case, power is computed only for the local network (that is up to the route connecting to the Internet).

The second operation allows estimating the energy that will be consumed by the network while handling a data transfer (for example when transferring a virtual machine image). For this, in addition to the flow descriptions, the input must include as well the data size (in bytes), the end points of the transfer, and a queuing model for each node. The transfer protocol can also be defined (for example http and ssh). These pieces of information allow calculating the data transfer time and the additional traffic load that will be expected to be applied to each participating network device to support the transfer, and from there the expected network energy consumption.

IV.3. Plug-in Interface

IV.3.1. Manager

The Manager oversees the communication between the core components of the plug-in and the data centre framework. For phase 1 purposes this component consists of two sub-components, the Monitor and the Controller.

The Monitor includes the interface for keeping the current model instance up to date and propagating requests coming from the Com components (see section IV.3.2.) to the Optimizer (see section IV.2.3.). It should be noted that the Monitor is responsible to ensure the integrity at any given time of the model representation. Since the latter can be modified at the same time by different components, the Monitor takes care of synchronizing all the concurrent update operations to preserve the consistency of the model over the time.

Table 10 shows the interface exposed by the Monitor.

allocateResource(Fit4GreenAllocationRequestType) : Fit4GreenAllocationResponseType
Allows a component to request directions on where to allocate new resources (for example on which server or which VM). The Monitor forwards the request to the Optimizer. It is invoked by the Com components.
Parameters:
allocationRequest the data structure containing the FIT4Green model instance and the specification of the resource to allocate
Returns:
a ResourceAllocationResponse object containing the results of the allocation request
requestGlobalOptimization(): void
Forwards global optimization request to the Optimizer
loadModel(String): boolean
Loads the FIT4Green model from an XML file and transforms it into an object hierarchy representation.
Parameters:
modelName the path to the XML model file

<p>Returns:</p> <p>true if success, false otherwise</p>
<p>updateNode(String, ICOM): boolean</p>
<p>Allows a Com component to update the FIT4Green model instance upon runtime modifications in the elements monitored by the Com. It works as a 'callback' operation. The Com invokes this method passing itself as a parameter. The Monitor, in turn, invokes an update method on the Com. This is needed for preserving the integrity of the object representation of the model, which can be updated by several components at the same time.</p> <p>Parameters:</p> <p>id the framework ID value of the node to update</p> <p>comObject the Com component which is asking for the update</p> <p>Returns:</p> <p>true if success, false otherwise</p>
<p>simpleUpdateNode(String, ComOperationCollector): boolean</p>
<p>Allows a Com component to update directly (without passing through the 'callback' mechanism) the FIT4Green model instance upon runtime modifications in the elements monitored by the Com.</p> <p>Parameters:</p> <p>id the framework ID value of the node to update</p> <p>operations collector of operations to be performed</p> <p>Returns:</p> <p>true if success, false otherwise</p>
<p>getModelCopy(): FIT4GreenType</p>
<p>Allows to get a deep copy of the FIT4Green model</p> <p>Returns:</p> <p>the object representation of the FIT4Green model</p>
<p>getMonitoredObjectsCopy(String): HashMap</p>
<p>Allows to get a deep copy of the subset of the FIT4Green model related to a com</p> <p>Parameters:</p> <p>comName the name of the Com</p> <p>Returns:</p> <p>the object representation of the FIT4Green model</p>
<p>getMonitoredObjectsMap(String): HashMap<String, ICom></p>
<p>Provides the set of all the nodes in the model which are handled by a given Com.</p> <p>Parameters:</p> <p>comName the name of the Com</p> <p>Returns:</p> <p>a map of all the objects handled by the 'comName' Com</p>
<p>logModel(): void</p>

Logs the XML representation of the current FIT4Green model instance

Table 10: Interface exposed by the Monitor.

The Controller includes the interface for propagating the actions suggested by the Optimizer to the corresponding Com component (see section IV.3.2.). Table 11 shows the interface exposed by the controller.

executeActionList(Fit4GreenActionRequest): boolean

Handles a set of requests and dispatch them to the responsible Com components in an optimal way.

Parameters:

actionRequest the action list suggested by the Optimizer

Returns:

true if successful, false otherwise

Table 11: Interface exposed by the Controller.

Figure 12 shows the most important control flows related to the Monitor and the Controller.

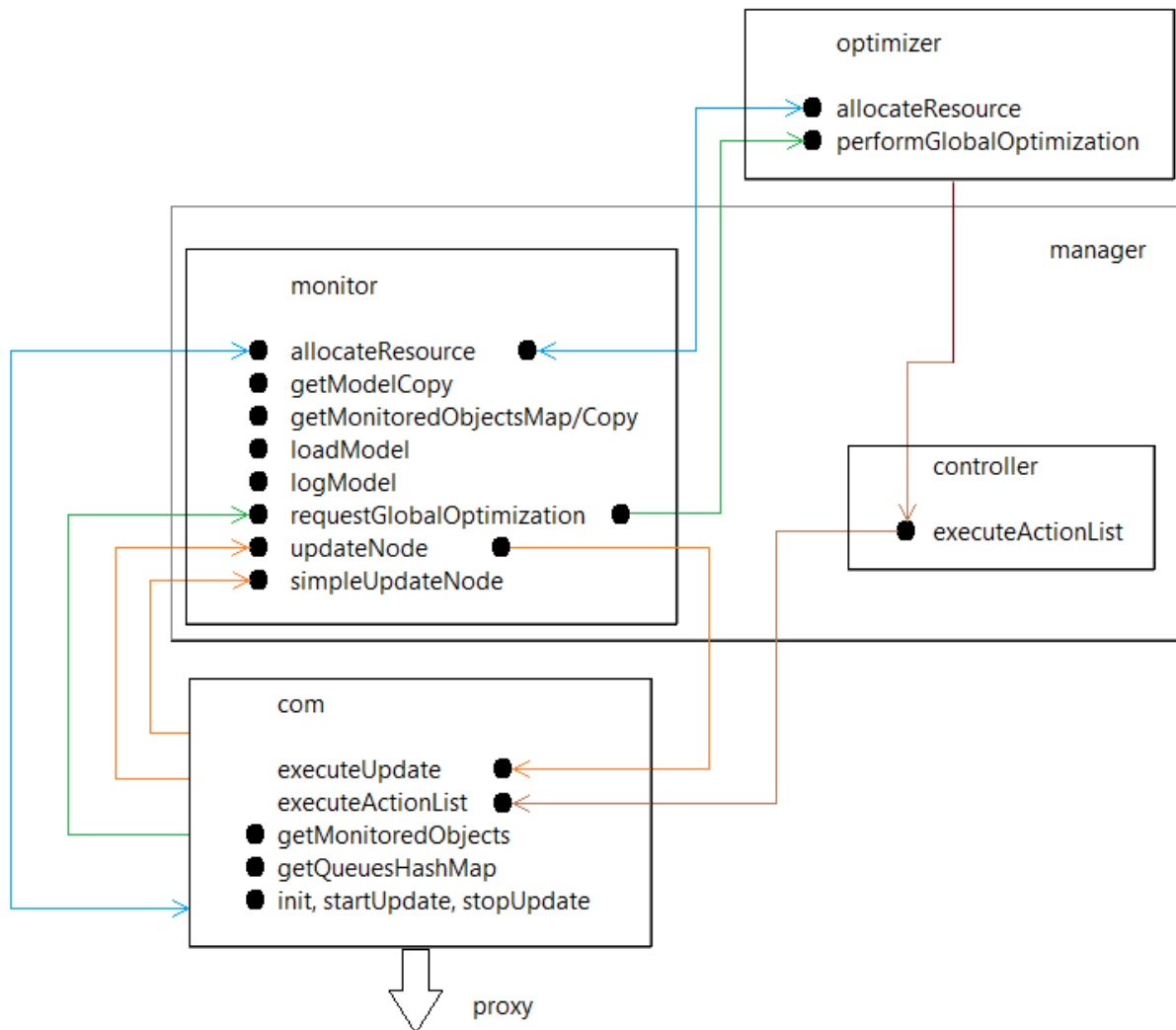


Figure 12: Control flows related to the Monitor and the Controller.

Currently, both sub-components are reactive, in the sense that they do not contain any logic to form and recognize patterns within the current model instance so as to make predictions

and take informed decisions. In the upcoming phases, the Manager is to be updated to accommodate for such behaviour.

IV.3.2. Communication (Com-Proxy Pair)

The FIT4Green plug-in can handle an extensible set of communication modules for interacting with different data centre automation and monitoring frameworks. For each communication module, a dedicated pair of components must be provided: a *Communication* component, deployed at the FIT4Green plug-in side, and a related *proxy* component, deployed at the data centre side.

The communication component and the related proxy enable the high-level interactions listed below.

- They allow the FIT4Green plug-in to perform direct operations on the specific data centre automation framework. Such operations are applied based on the directions stated by the Optimization engine and dispatched by the Controller component.
- They allow the specific data centre monitoring framework to transmit status data towards the FIT4Green plug-in, updating the related entries in the model instance. Such data is then used by the Optimization engine, to calculate the most appropriate actions needed for minimizing the overall energy consumption.
- They allow the specific data centre automation framework to ask the FIT4Green plug-in for a global optimization cycle, or for directions on where to allocate new resources within the data centre.

All the communication components implement the same `org.f4g.com.ICom` and `org.f4g.com.IComOperationSet` interfaces, and might extend the abstract class `org.f4g.com.AbstractCom`, which provides the implementation of a set of basic functionalities shared by all of them.

The specific description of each communication component currently available, together with their associated proxy, is provided in the next subsections.

Here follows the description of the interfaces introduced above.

ICom interface

The ICom interface defines all the methods that must be implemented by a 'communication' component, as shown in Table 12.

executeUpdate(String, Object): boolean
<p>Performs a 'complex update' operation on the model, in the sense that it not only updates existing values, but also modifies the model structure by adding/removing nodes. A communication component that needs a complex update, invokes the 'updateNode()' method on the Monitor, which in turns invokes this method as a 'callback' operation. This strategy is needed to grant consistency in case of concurrent update operations on the model coming from different communication modules. The Monitor is the collector of the requests and provides guarantees on the consistency of the model by synchronizing them. The Monitor passes a node of the model into the object parameter. The Com retrieves a set of update operations queued for this node and applies them to it.</p> <p>Parameters:</p> <p>key the key of the mapping for the Com object</p> <p>obj a node in the FIT4Green model</p> <p>Returns:</p>

<p>true if successful, false otherwise</p>
<p>init(String, IMonitor): boolean</p>
<p>Initializes the component. Retrieves from the Monitor a list of the node objects monitored by the component. Creates a new map with the same keys, but for each key (meaning for each node) a queue is created. The queue will contain the list of pending updates to be performed.</p> <p>Parameters:</p> <p>name the</p> <p>monitor the</p> <p>Returns:</p> <p>true if init with no errors, false otherwise</p>
<p>startUpdate(): boolean</p>
<p>Starts up the control loop of the Com, for the monitoring of the data and the update of the model.</p> <p>Returns:</p> <p>true if starts up with no errors, false otherwise</p>
<p>stopUpdate(): boolean</p>
<p>Stops the control loop of the Com.</p> <p>Returns: true if stops with no errors, false otherwise</p>
<p>dispose(): boolean</p>
<p>Called by the core component responsible for starting up and shutting down the FIT4Green plug-in. It must implement all the operations needed to dispose the component in a clean way (such as stopping dependent threads, closing connections, sockets, file handlers, and so on).</p> <p>Returns:</p> <p>true if dispose with no errors, false otherwise</p>
<p>executeActionList(ArrayList): boolean</p>
<p>Executes a list of actions on behalf of the Controller. Each action is the object representation of an xml data structure, which defines a specific operation to be performed on a data centre automation framework (together with all the parameters needed). The XML schema for each operation has been defined in WP4. Every operation is mapped into the IComOperationSet interface (defined below).</p> <p>Parameters:</p> <p>actionRequest the data structure representing the list of actions to perform</p> <p>Returns:</p> <p>true if successful, false otherwise</p>
<p>getMonitoredObjects(): HashMap</p>
<p>Provides the set of all the nodes in the model which are handled by the Com.</p> <p>Returns:</p> <p>the set of the nodes in the model handled by the Com</p>
<p>getQueuesHashMap(): HashMap</p>

For each element handled by the com, there is a ConcurrentLinkedQueue structure, acting as a collector for the update operations to be performed on the model for that element. When a complex update operation is invoked on the Monitor for an element, the corresponding queue is evaluated and the related update operations executed on the model

Returns:

the HashMap containing the mapping <element, ConcurrentLinkedQueue>

Table 12: Interface exposed by a Com component.

IComOperationSet interface

This interface represents all the possible operations allowed on a Com component. Such operation set has been entirely defined in WP4.

Every component must implement it. If a method is not supported, an error response is returned. The operations listed in Table 13 are self-explanatory.

<code>powerOn(PowerOnActionType): boolean</code>
<code>powerOff(PowerOffActionType): boolean</code>
<code>sleep(SleepActionType): boolean</code>
<code>liveMigrate(LiveMigrateActionType): boolean</code>
<code>moveVm(MoveVMActionType): boolean</code>
<code>pause(PauseActionType): boolean</code>
<code>resume(ResumeActionType): boolean</code>
<code>boot(): boolean</code>
<code>suspend(): boolean</code>
<code>deactivate(): boolean</code>
<code>activate(): boolean</code>
<code>setFrequencyValues(): boolean</code>
<code>shutdown(): boolean</code>

Table 13: Operations allowed on a Com component.

IV.3.2.a. Traditional Computing - ENI

ENI Com is the component that manages traditional data centre at ENI facilities. It communicates with ENI servers: updates the model with monitoring data and propagates the suggested actions to the servers.

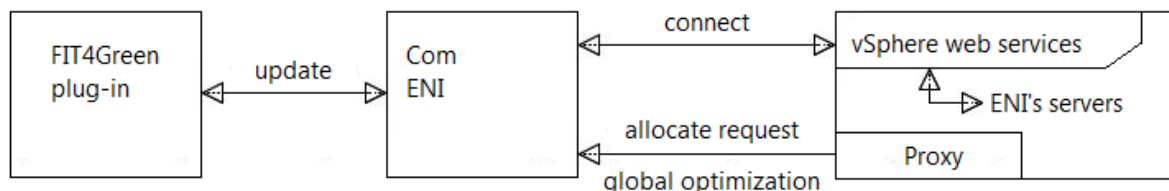


Figure 13: ENI Com component and its connection to ENI's servers.

Connection to FIT4Green

ENI Com extends `AbstractCom`; the way that the suggested by the Optimizer actions are further propagated is specified within `executeActionList` in `AbstractCom`. ENI Com

overrides these actions and sends them directly to ENI's servers. It also performs a monitor loop until component itself is stopped, connects to servers and updates the FIT4green model by creating or deleting VMs as needed in the model.

Connection to ENI's

ENI's servers run VMWare vSphere to administrate VMs; this hypervisor provides web services to access vSphere components.

ENI Com connects to the vSphere web services and performs the necessary operations to optimize, such as migrate a VM or power off a node. It also retrieves the necessary data to update the model, such as number of hosts, VMs available running on the hosts, memory used, and so on.

The Proxy component deployed on ENI's servers sends global optimization requests and allocation requests for resources like VMs to the FIT4Green plug-in via the ENI Com.

IV.3.2.b. Cloud Computing - HP

The FIT4Green plug-in can handle a Cloud Computing environment by means of a dedicated pair of components: the *Cloud Computing Communication (CCC)* component, deployed at the FIT4Green plug-in side (COM), and a related *proxy* component, deployed at the Cloud Computing environment side.

The CCC component and the related proxy enable the high-level interactions (see section IV.3.2.) as all the other FIT4Green communication/proxy component pairs.

Cloud Computing Communication (CCC) component

The communication between the CCC component and the proxy component is based on commands and data exchanged over the SSH network protocol, on a dedicated port defined at configuration level.

The CCC component is started during the initialization phase of the FIT4Green plug-in. Upon initialization, leveraging the utilities provided by the Monitor component, the CCC component retrieves from the model instance (which reflects the configuration of all the data centres managed by the FIT4Green plug-in) the subset of elements under its responsibility (for example the servers or the enclosures available in the Cloud Computing environment, together with their subcomponents). Then, it enters in a loop lasting until the component itself is stopped by the plug-in; during the loop, at given intervals defined at configuration level, it opens a communication session with the proxy deployed on the Cloud environment to query status information. The data retrieved are then transmitted to the Monitor component as argument of an update operation, whose effect is the update of the model instance with the most up-to-date values available. In case such values correspond to items already present in the current model instance, the update operation is called 'simple update', and it is performed by invoking the `simpleUpdateNode()` method on the Monitor. In case a virtual machine is powered on or off, the update operation consists of a structural modification of the model instance, where a related set of nodes are added or removed. This operation is called 'complex update' and is performed invoking the `updateNode()` method on the Monitor, which in turn invokes the `executeUpdate()` method on the Com. Such method is based on a callback mechanism and define in the ICom interface (see section IV.3.2.).

In addition to the loop for the monitoring of the Cloud Computing environment, the CCC component also runs a SSH server on a dedicated port. This mechanism, as explained later, allows the proxy component to communicate proactively with the CCC.

Moreover, the CCC provides means for performing on-demand operations on the Cloud Computing environment (via the proxy), on behalf of the Controller component and based on the directions of the Optimization engine.

Cloud Computing proxy component

The proxy component acts as the interface towards the native 'Monitoring and Automation Framework' of the Cloud Computing environment. The proxy relies on a set of scripts that allow:

- Querying the Monitoring framework of the Cloud environment for the set of resources available and for their status. As an example, a typical set of data retrieved is the number of virtual machines running on a node, details related to the number of their CPUs, available memory, current CPU load, and so on.
- Sending specific commands to the Automation framework of the Cloud Computing environment, to perform operations such as powering on / off a virtual machine on a given node

In addition to the operations mentioned above, which are triggered by requests coming from the CCC level, the proxy can also proactively interact with the CCC by sending a request for an overall global optimization or a *request for resource allocation directions*.

A request for an overall optimization can be helpful in specific circumstances, in particular when one or more virtual machines are powered off in a node. Actually, the disappearance of virtual machines from the model instance, once detected by the plug-in, would likely trigger a global optimization because a relevant amount of resources has been made available. By triggering an optimization request proactively, the proxy allows to inform the plug-in immediately, thus minimizing the time during which the energy consumption is not optimized.

A request for allocation directions allows the Cloud Computing environment to ask the FIT4Green plug-in for directions on where (for example on which physical server) to allocate new resources. Typically, this is helpful when one or more virtual machines are going to be started in the data centre and different physical servers are available for hosting them.

The proactive communication is performed by sending a specific command over the SSH protocol to the CCC SSH server.

Figure 14 depicts the internals of the Cloud Computing proxy component.

The blue items represent standard products or software tools available in the HP Cloud Computing environment, while the yellow items represent custom components developed for FIT4Green.

The upper flow of the figure ('MONITOR' action) shows how the proxy retrieves the information related to the system load and to the power consumption from the Monitoring Framework. The collection of such monitoring data is enabled by CollectD, a standard OS daemon that collects system performance statistics periodically and is capable of storing them in a special format (RRD files), which can then be easily accessed via a set of dedicated, standard tools (CollectD tools). Driven by a configuration file, the CollectD daemon queries in a loop a set of 'Agents', which are typically scripts capable of retrieving performance information for specific system components. For the Cloud Computing proxy, we handle the following agents:

- A 'system agent', which retrieves statistical information related to the performance of physical nodes
- A 'libvirt' agent, which retrieves statistical information related to the performance of virtual machines running on a set of nodes
- A 'ilo agent' which retrieves statistical information on the power consumption of the associated hardware devices

All the data collected by the agents is stored in the 'CollectD rrd' repository and is made available to the proxy by means of the CollectD tools.

When the CCC sends a specific SSH command for getting performance data to the proxy component, the latter invokes the CollectD tools, which in turn query the RRD repository fed by the agents, and return the data back to the proxy. Such data are then given back to the CCC, which parses them and, if needed, updates the FIT4Green runtime model.

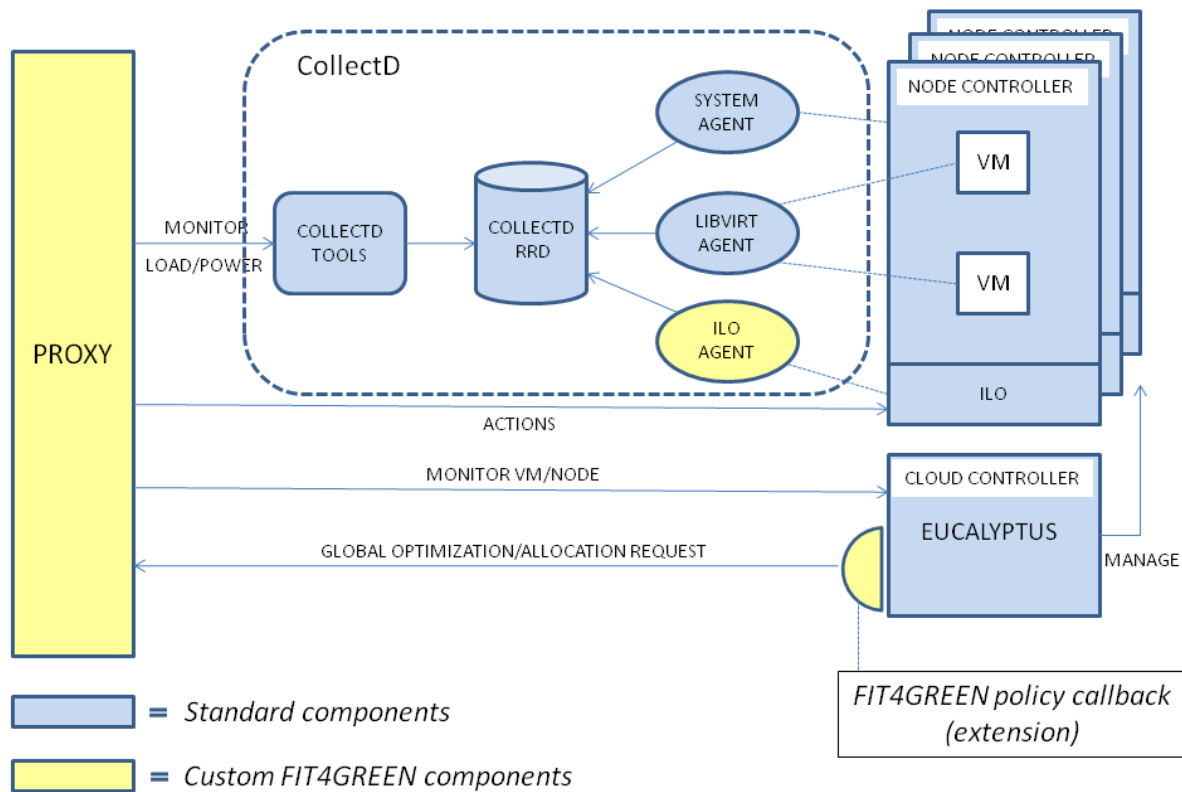


Figure 14: Proxy component for the Cloud Computing environment.

The ‘ACTIONS’ arrow represents direct actions that must be performed on the environment, such as starting up or shutting down virtual machines. In this case, the proxy interact directly with the ILO (ILO is a special interface that makes it possible to perform activities on a server from a remote location) of the target physical node.

The ‘MONITOR VM/NODE’ arrow represents an action performed by the proxy on the ‘EUCALYPTUS’ cloud controller (which manages the whole cloud environment), to retrieve information on:

- The physical nodes active in the Cloud Computing environment
- The virtual machines running on each physical node

Finally, the EUCALYPTUS cloud controller has been extended with a FIT4Green policy call-back mechanism, which allows to proactively sending requests to the FIT4Green plug-in (forwarded by the proxy to the CCC) for:

- Performing a global optimization (for example after a virtual machine shutdown)
- Getting directions on where to allocate new virtual machines (for example when a new virtual machine must be started)

The policy call-back extension is represented by the yellow semicircle close to the EUCALYPTUS cloud controller, while the ‘GLOBAL OPTIMIZATION/ALLOCATION

REQUEST' arrow represents the requests performed by the cloud controller towards the proxy.

IV.3.2.c. Supercomputing - JSC

For the supercomputing environment FIT4Green implements a green cluster job scheduler which allocates jobs to cluster nodes in an energy-efficient way. For the first test phase the strategy for power savings is based on setting nodes in sleep/power-off mode in case there should be no jobs queued which could make use of them.

The FIT4Green engine runs (currently) on a dedicated server located in JSC's DMZ (DeMilitarized Zone) allowing for remote access to the project partners. The engine includes a model of the configured environment, a monitor, an optimizer, a controller, and a communicator plug-in for each data centre. The proxy interface to the cluster framework is installed on the local frontend systems.

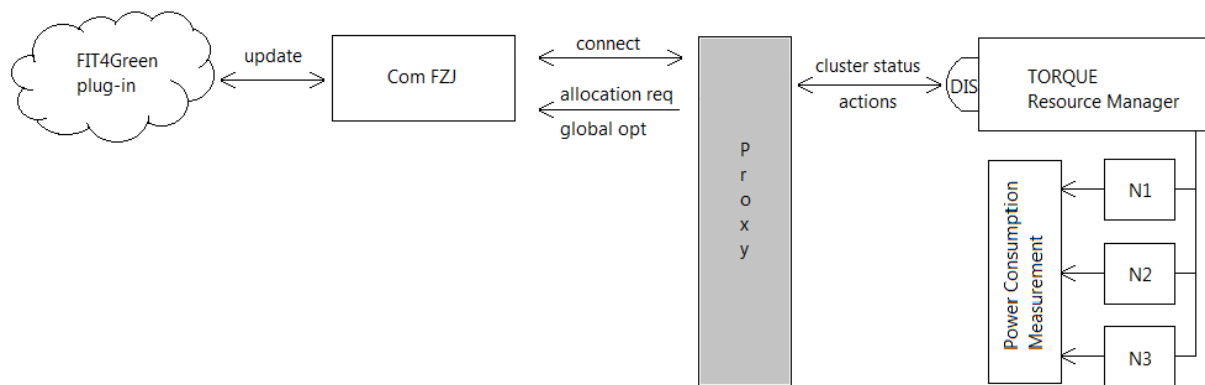


Figure 15: Com and Proxy component at JSC.

When started, the ComFzj plug-in polls the monitor for the latest status. Then it performs a status update on a regular basis (every 30 seconds). To this end, a connection to the proxy is established where the current information about the jobs, the nodes, and the queues is collected. In case the cluster status has changed the FIT4Green monitor is invoked and updated as well as a new global optimization is requested for the resource management of the cluster. The FIT4Green response to this request consists in a list of actions that should be executed on the cluster, for example schedule a job to a specific node, or power off some nodes.

The proxy daemon on the cluster frontend is responsible for providing the latest cluster status to the ComFzj plug-in. The communication between proxy and the resource manager Torque is based on the DIS [2] software interface, an ASCII-coded, message-based protocol allowing for a consistent view to the cluster resources.

The test environment at JSC comprises two types of Linux clusters:

- Juggle: 1 front-end, 4 compute nodes
 - dual AMD double-core, AMD Opteron F 2216 2.4 GHz,
 - 8 GB memory, InfiniBand, Gigabit Ethernet
 - Operating mode: batch (TORQUE)
- Jufit: 1 front-end, 2 compute nodes
 - dual Intel six-core, Intel Westmere X5660 2.80GHz,
 - 24 GB memory, InfiniBand, Gigabit Ethernet
 - Operating mode: batch (TORQUE)

The Juggle hardware has been in use for over 3 years whereas the Jufit nodes represent state-of-the-art hardware. All nodes are connected to an outlet allowing for power-measurement for each single node.

These small clusters shall represent a typical usage of the larger supercomputer systems. So, a workload generator has been provided which simulates the submission of typical user jobs (CPU, memory, and IO bound). The cluster resource manager holds all jobs in a queue.

IV.3.2.d. Network - ICL

The Communicator Network module deals with monitoring the network and extracts the required information for the use of the Optimizer. We use the Simple Network Management Protocol (SNMP) for the network monitoring, which is a component of the TCP/IP. In this phase, the observed information are send and receive traffic from each networking device in the data centre, that is, routers, switches and the embedded network devices in other nodes. The requested information is defined by Management Information Bases (MIBs). Hence, traffic in/out for the uni-cast, multi-cast, and broadcast communications at each network node can be queried using its corresponding MIB. Therefore, the MIBs that can retrieve all the above information include: `ifInUcastPkts`, `ifInMulticastPkts`, `ifInBroadcastPkts`, `ifOutUcastPkts`, `ifOutMulticastPkts`, and `ifOutBroadcastPkts`. The traffic in/out is later used for the calculation of power associated to data transmission, which is also utilised by the Optimiser to examine the cost of relocating certain VM.

IV.4. Web Applications

IV.4.1. InitServlet

As described in section III.1. , the FIT4Green plug-in is to be delivered as a self-contained virtual machine image, containing the entire software infrastructure needed to run it. When the virtual machine image is launched, a Tomcat server containing a set of web applications needed for controlling the FIT4Green services is automatically started up. The InitServlet is one of these web applications and plays a special role: depending on the configuration, it can automatically initialize and launch the FIT4Green plug-in as soon as the Tomcat server starts up. This allows having the plug-in up and running as soon as the virtual machine has completed the initialization phase, without the need of manual actions from an operator.

The possibility of starting up the plug-in automatically by means of the InitServlet can be set at configuration level in the main FIT4Green configuration file:

```
[ $TOMCAT_HOME ] / common / classes / config / f4gconfig.properties
```

by setting the `automaticStartup` property to true or false, as defined in the following excerpt from the `f4gconfig.properties`:

```
#automaticStartup: defines if the f4g plugin must be started up
#automatically when Tomcat is started
#   - set to 'true' for automatic startup,
#   - set to 'false' for manual startup,
automaticStartup=true
```

The InitServlet is a simple servlet whose actions are limited to:

- loading the global FIT4Green plug-in configuration file `f4gconfig.properties`
- verifying if the property `automaticStartup` is set to true
- (provided that automatic startup is set to true) getting an instance of the `IMain` interface (which is the entry point to the FIT4Green plug-in) and invoking the dedicated methods for starting up the plug-in

The InitServlet is configured to be loaded automatically at start up of the Tomcat server. This is accomplished by setting the following property in the standard `web-xml` servlet descriptor file:

```
<load-on-startup>1</load-on-startup>
```

If the servlet is configured for manual startup, that is `'automaticStartup=false'`, then the plug-in must be started explicitly by an operator by using the FIT4Green web UI (see section IV.4.2.).

IV.4.2. Web User Interface

The FIT4Green UI is a web based interface to manage the FIT4Green plug-in. The interface contains three tab pages.

- The *Status* page, Figure 16, contains buttons to start, stop, and optimize the system, and shows the current state (stopped or running). It also shows a list with the action history.
- The *Configuration* page, Figure 17, lists all configuration files, such as models, SLAs, data centre constraints and workload, and plug-in and Com properties files. The files can be viewed, edited, created, and deleted. The configuration files can be edited in an XML editor, while the properties files are simple text files.
- The *Help* page is to contain documentation for the operators on how to use the plug-in.



Figure 16: Status tab of the FIT4Green Web UI.

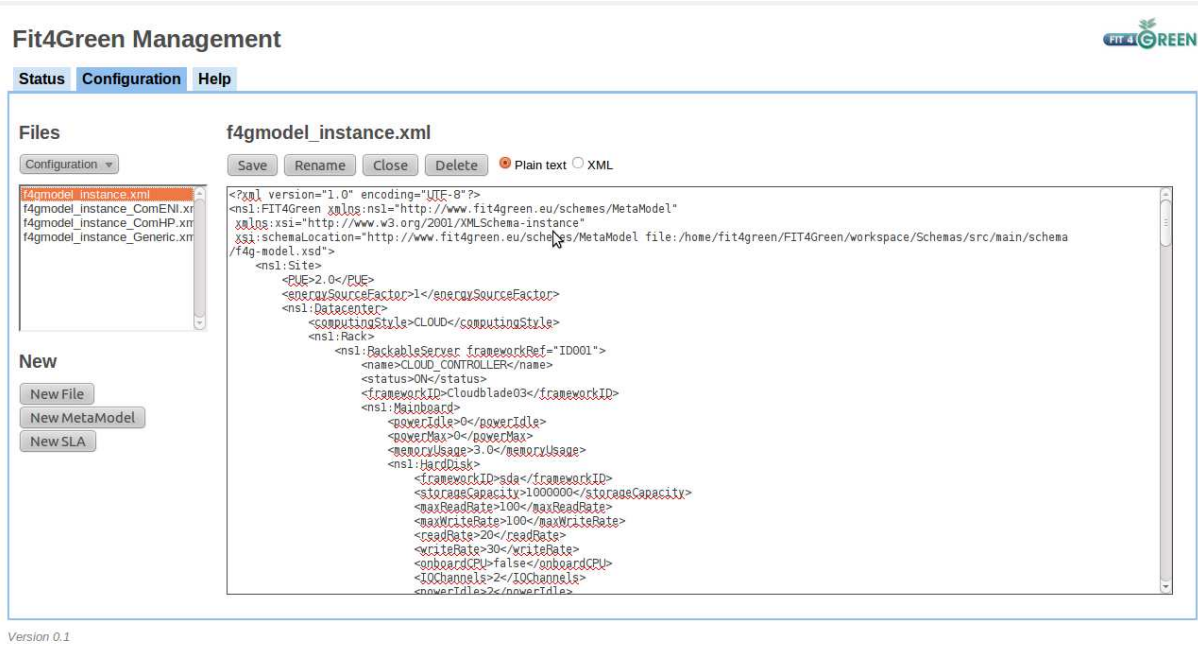


Figure 17: Configuration tab of the FIT4Green UI.

The source code consists of one Eclipse project, named `f4gGui`. It is a Google Web Toolkit (GWT) project, and has both client and server side code. Its schematic overview is shown in Figure 18.

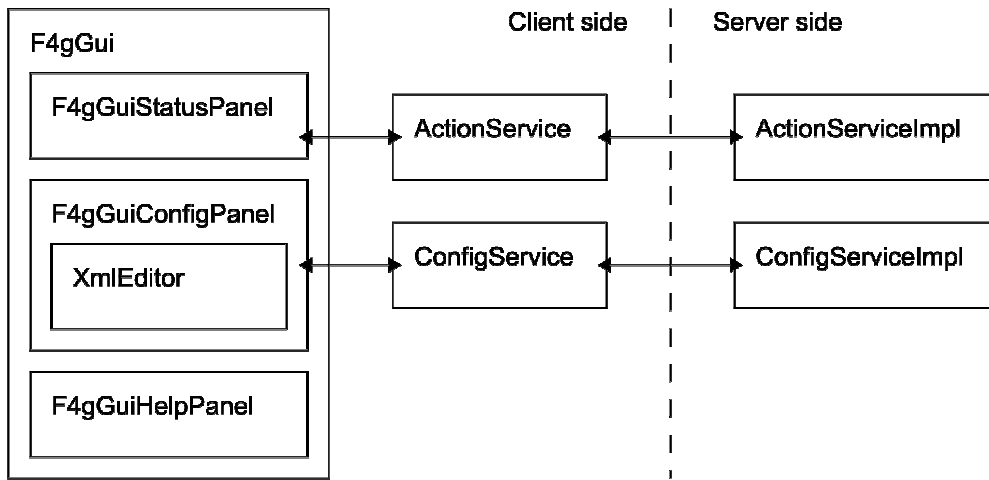


Figure 18: Schematic overview of the FIT4Green UI.

Server side

Two servlets are running on the server side: `ActionServiceImpl` and `ConfigServiceImpl`.

The `ActionServiceImpl` servlet communicates with the FIT4Green plug-in, and is able to send the following actions to the main instance: start-up, shutdown, and optimize.

The `ConfigServiceImpl` servlet is capable of reading, writing, creating and deleting files in the configured folders. The configured folders - currently Configuration and Properties - are retrieved from the properties file `f4ggui.properties`, which contains a list with directory titles and the location of the directories. The configuration file is located in the Tomcat directory:

[\$TOMCAT_HOME] / common / classes / config

Client side

The entry point of the web interface is the class `F4gGui`; this class is loaded when the UI is called and creates a page with a title and a tab control. The tab control contains three pages, which have their own source file: `F4gGuiStatusPanel`, `F4gGuiConfigPanel`, and `F4gGuiHelpPanel`.

The `F4gGuiStatusPanel` contains buttons to start, stop and optimize the system, and lists the action history.

The `F4gGuiConfigPanel` contains a list with directories, a list with the files in these directories, buttons to create, delete, edit, save, and rename files, and an editor where the contents of a file are displayed and can be edited.

The files can be displayed in two ways: in a plain text area, or in an XML editor. The XML editor can only be used for XML configuration files. The used XML editor is an open source Java Applet called `xamplet.jar` [3]. This applet is embedded in the UI via the class `XmlEditor`. The XML editor can only edit XML files with a corresponding XSD schema. The correct XSD is read from an XML file's namespace. The available XSD schemas are located on the client side in the directory `schemas`.

The `F4gGuiHelpPanel` is currently empty, but will contain documentation for the operators on how to use the FIT4Green plug-in and its UI.

The client side communicates with the two servlets via RPC calls (the default GWT way), defined in the classes `ActionService`, `ActionServiceAsync`, `ConfigService`, and `ConfigServiceAsync`.

V. SUMMARY AND OUTLOOK

In this document we have discussed the design and implementation of the energy control plug-in for single-site data centres. The FIT4Green plug-in can be thought of as a set of software components that adds energy management capabilities to a data centre by interfacing with - *being plugged into* - the management, both controlling and monitoring, and automation system - the so-called *framework* - of the data centre.

During the first test phase our plug-in is pretty reactive, for example the plug-in:

- uses the sample rate available to the data centre monitoring tools in order to keep the model up-to-date
- has no knowledge (yet) over the importance of changes in parameter's value nor different sample rates
- propagates all allocation requests
- may request global optimization in case the data centre operator has requested it through the web UI or based on some crude rule-of-thumbs such as "a VM has been un-deployed"; however, in reality has no substantial knowledge over when would be the best time to request a global optimization.

Work on phase 2 is to deal with the above mentioned shortcomings of the single-site data centre plug-in as well as to extend the solution to federated-site data centres.

VI. REFERENCES

- [1] **Appleton, Brad.** A Software Design Specification Template. [Online] 1997. <http://www.cmcrossroads.com/bradapp/docs/sdd.html>.
- [2] DIS Encoded Batch Requests. *Portable Batch System*. [Online] 5 September 2008. [Cited: 14 January 2011.] http://www.fhi-berlin.mpg.de/th/locserv/software_all/PBS/full/ERS-194.html.
- [3] **Golubov, Felix.** Java XML Editor - GUI Generator. [Online] 3 July 2004. [Cited: 14 January 2011.] <http://www.felixgolubov.com/XMLEditor/>.
- [4] **Sommerville, Ian.** *Software requirements*. England : Pearson Education Limited, 2007. pp. 117 - 140.

APPENDIX A: D5.1 ANNEXES

Table 14 lists the annexes accompanying deliverable D5.1. As the needs of the project evolve, these documents are updated accordingly.

Annex #	Title	Description
1	Coding Conventions	Lists the Java coding conventions agreed upon in the FIT4Green project
2	Configuration Management Plan	Presents the FIT4Green configuration management policies, describing mainly the best practices for Subversion usage, along with the FIT4Green own policies and rules
3	Project Configuration Description	Describes the FIT4Green development environment and provides guidelines to set it up and running
4	Languages and Tools	Provides information regarding the programming languages as well as the simulation and modelling tools used within FIT4Green
5	Release Notes V1.0	Lists features and known issues of the current release, that is, V1.0

Table 14: Annexes accompanying D5.1

