Measuring Green Software Enginnering Monitoring Software Energy Consumption

In the MEASURE ITEA 3 Project

Alessandra Bagnato, Marcos Aurélio Almeida da Silva, Antonin Abherve
Softeam Research & Development Division
Paris, France
alessandra.bagnato@softeam.fr

Jérôme Rocheteau, Claire-Lise Pihery, Pierre Mabit
Institut Catholique d'Arts et Métiers
35 avenue du champ de Manœuvres, Carquefou, France
jerome.rocheteau@icam.fr

***Abstract*— This paper highlights the benefits within the Green Computing metrics measurement context from the MEASURE ITEA 3 project (Measuring Software Engineering) Project French cluster. It presents the Structured Metrics Meta-model (SMM) used as a standardized language and its implementation within the Softeam's Modelio modelling and ICAM's EMIT, a set of tools able to provide software power and energy measurements, its result mapping into SMM and the proof of its interoperability with Modelio and with all MEASURE partner tools in the future.**

Keywords—Green Computing, Metrics, Software Sustainability, Green Measures.

I. Introduction

The goal of the MEASURE ITEA 3 (see [1] and [2]) is to increase the quality and efficiency, and lower the costs and time-to-market of software products in Europe. MEASURE (Measuring Software Engineering) project (see [1] and [2]) will provide a toolset for future projects to properly measure their quality and their impact and in particular to develop methods and tools for analysing the big data produced by the continuous measurement and the advanced analytics of the measurement data enabled by the project.

The project will specifically focus on Green Computing metrics as ICT carbon emissions increase continually these past years. Therefore, an emerging trend in software engineering consists in building energy efficient software. Alongside investigating what is a performant or a maintainable software for instance, the MEASURE project aims at defining what is an energy-efficient software and how to assert such a statement.

In this paper we analyse the green metrics contribution carried out by the project. This paper focus in particular on the MEASURE's EMIT tools. The objective of these tools is to monitor software energy consumption.

The Structured Metrics Meta-model (SMM) is used in order to formally define measures and to represent measurements related to these measures. Such a standardized meta-model enables tool interoperability as their common exchange language. It is presented in Structured Metrics Meta-Model (SMM). Section 3 presents some Green Computing metrics proposed throughout the MEASURE project and explains the formalization of one example within SMM while Sections IV provides an extract of the library of green measures defined using the SMM standard. Sections V to XI are dedicated to the MEASURE EMIT power measurement toolset, its components, its data model, its functionalities and its measurement mapping into SMM. Some experiments are providing in order to asset its reliability. Section XII concludes the paper.

II. Structured Metrics Meta-Model (SMM)

Most software system properties can be quantified with the application measurement processes. OMG's Structured Metrics Meta-Model (SMM) supports the meta-model agnostic definition of those measurement processes.

The SMM specification defines a meta-model for representing measurement information related to any model structured information with an initial focus on software, its operation, and its design. SMM is an extensible meta-model for exchanging both measures and measurement information concerning artefacts contained or expressed by structured models, such as MOF.

SMM includes elements representing the concepts needed to express a wide range of diversified measures. The specification does include a minimal library of software measures, but it is not asserting that the listed measures constitute standards themselves; these are supplied simply as non-normative examples.

SMM is a specification for the definition of measures and the representation of their measurement results. The measure definitions make up the library of measures and that serves to establish the specification upon which all of the measurements will be based.

SMM is part of the Architecture Driven Modernization (ADM) roadmap and fulfils the metric needs of the ADM roadmap scenarios as well as other information technology scenarios.

SMM specifies the representation of measures without detailing the representation of the entities measured. SMM anticipates that those entities are represented in other OMG meta-models. Measures of software artefacts or their features that are defined within the SMM, the Knowledge Discovery Metamodel (KDM), the Abstract Syntax Tree Metamodel (ASTM), another ADM roadmap meta-model, or another OMG meta-model may arise as:

- Counts. (Lines of code measures exemplify the mechanism.)
- Direct applications of named measurements. (One such named measure is Cyclomatic Complexity.)
- Simple algebraic change of calibration of already defined numeric measures (e.g., the translation to 'choice points' from Cyclomatic complexity).
- Simple algebraic aggregations of numeric artifact features, including other measures, over sets of software artifacts.
- Simple range-based grading or classification of already defined numeric measures. (Cyclomatic reliable/unreliable quadrants are one such grading.)
- Qualitative evaluations where the range of evaluations can be mapped to a linear order.

Useful metrics must go beyond static (or dynamic) code analysis and technical performance to include factors related to information utility and acceptance of the system by the organization(s) participating in an enterprise. To be objective and repeatable, such metrics need to be based on technical characteristics of the system. Given a meta-model representation of such characteristics, the SMM will facilitate the exchange of such measures [3]. Consistent with other models defined by OMG, the SMM is defined using the MOF meta-modeling language. As such it will have a standard XML based representation presented by XMI.

Consequently, the exchange of metrics defined by SMM will be in the XMI. These models will, similarly, be compatible with MOF repositories for storage and retrieval by various tools. [3].

III. MEASURE Green Computing Metrics

Green Computing metrics have been proposed in this past decade. However, they mainly focus on hardware, based on the power supply of such components and the carbon footprint of the life cycle of these products. The few that tackle software, strongly restrict application domains, for instance to data centres, virtual machines or mobile technology only. Furthermore, these approaches have been based on power consumption at runtime, e.g., during software execution according to some usage scenarios or use cases. Indeed, all Green Computing metrics target users of the devices or applications. As such, metrics are relevant for choosing products with the smallest energy consumption and answering the question: "which software products are green". These metrics do not target software engineers and do not answer the question "how to produce greener software" [5]. The MEASURE project will strengthen the concept of Green IT, as well as the research regarding energy and smart systems. Within this section an overview of the main green metrics that will be tackled. The modeling formalisms supported by MEASURE will also investigate the existing correlation between software development measurements and the quality of end-user experience providing cross-metrics feedback very much needed in industry.

The following table shows the measure green metrics currently under development, they purpose is to provide input for finding relevant elements during the Testing phase, to challenge the energy consumption of a given code against computing-equivalent codes. The library takes into account POWER, RAM, CPU, VOLTAGE, INTENSITY and FACTOR.

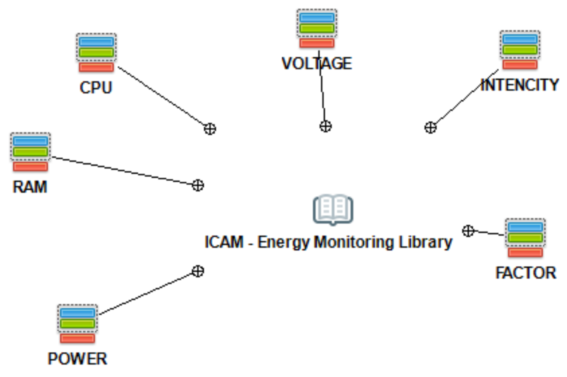


Figure 1 : MEASURE Energy Monitoring Library

TABLE I. OVERVIEW OF MEASURE GREEN METRICS.

Green Measure name	**Purpose**		
	Purpose	*EMIT measuring tool*	**SMM Design**
Energy Efficiency	Statistical distribution of measurands on energy efficiency classes from A to F based on their energy consumption.	.	no
Average CPU	Average CPU load of a given measurand	EMIT Sigar or Oshi	yes
Standard Deviation CPU	Standard deviation of the CPU load of a given measurand	EMIT Sigar or Oshi	yes
Minimum CPU	Minimum CPU load of a given measurand	EMIT Sigar or Oshi	yes
Maximum CPU	Maximum CPU load of a given measurand	EMIT Sigar or Oshi	yes
Average RAM	Average RAM usage of a given measurand	EMIT Sigar or Oshi	yes
Standard Deviation RAM	Standard deviation of tha RAM usage of a given measurand	EMIT Sigar or Oshi	yes
Minimum RAM	Minimum RAM usage of a given measurand	EMIT Sigar or Oshi	yes
Maximum RAM	Maximum RAM usage of a given measurand	EMIT Sigar or Oshi	yes

Green Measure name	**Purpose**		
	Purpose	*EMIT measuring tool*	**SMM Design**
Average Power	Average power of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Standard Deviation Power	Standard deviation of the power of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Minimum Power	Minimum power of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Maximun Power	Maximum power of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Average Voltage	Average voltage of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Standard Deviation Voltage	Standard deviation of the voltage of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Minimum Voltage	Minimum voltage of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Maximum Voltage	Maximum voltage of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Average Intensity	Average instensity of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Standard Deviation Intensity	Standard deviation of the instensity of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Minimum Intensity	Minimum instensity of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Maximum Intensity	Maximum instensity of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Average Factor	Average power factor of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Standard Deviation Factor	Standard deviation of the power factor of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Minimum Factor	Minimum power factor of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes
Maximum Factor	Maximum power factor of a given measurand	Power_GUI, Arduino_PO, Peaktech_PA	yes

Section IV describe the currently available library of Green measures specified and modelled with the structured metrics metamodel

The examples in that section are the first building block of the MEASURE project tool chain, the Modelio modeling tool enabled with the SMM Module developed based on the Structured Metrics Metamodel OMG Standard in Modelio's open source distribution to allow the specification of metrics and in particular of green metrics and the exchange of metrics within tools using XMI.

IV. Library of Green measures defined using the SMM Standard

This excerpt of our green measure example library shows the CPU related metrics including average CPU, standard deviation CPU minimum CPU or maximum CPU, defining the % of CPU used in a Modelio SMM model.

SMM Measures are gathered according to the type of measure that they are related to. For instance, the figure 2 and the figure 3 illustrate the group of measures around the CPU load of measurands. In fact, a SMM direct measure exists in that group and is linked with a unit of measure. However, no one SMM direct measurement will be inserted into the SMM Library of Green Measures as such measurements correspond to raw measurements and provide no information.

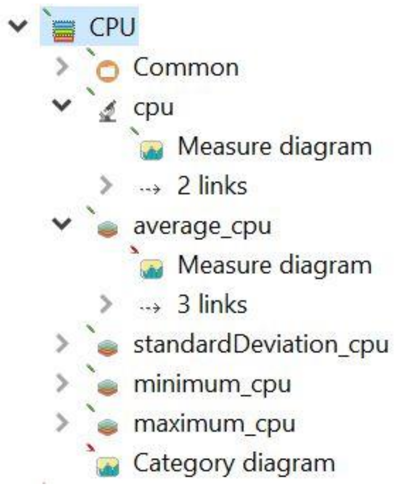


Figure 2: CPU metrics model modelled in Modelio SMM

Information is provided by analysis of these raw measurements, The SMM CPU direct measure is then used for defining some SMM collective measures that stands for statistical measures with a SMM accumulator that ranges over average, standard deviation, minim, maximum, etc.

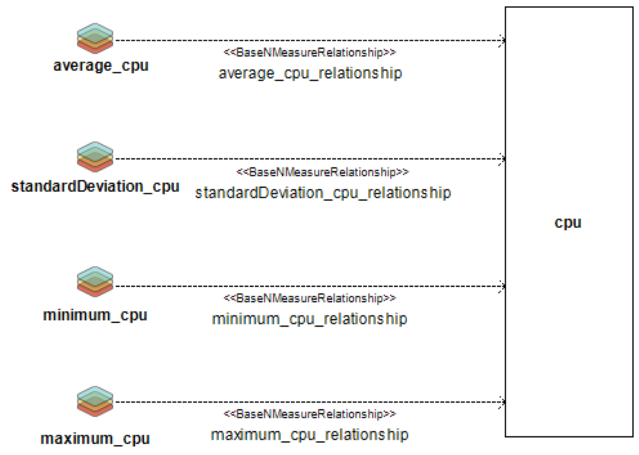

average_cpu
<<BaseNMeasureRelationship>>
average_cpu_relationship
standardDeviation_cpu
<<BaseNMeasureRelationship>>
standardDeviation_cpu_relationship
minimum_cpu
<<BaseNMeasureRelationship>>
minimum_cpu_relationship
maximum_cpu
<<BaseNMeasureRelationship>>
maximum_cpu_relationship
cpu

Figure 3: CPU Category Diagram, CPU associated metrics modelled in Modelio SMM

These SMM collective measures are then linked to a SMM measure binary relationship as its source and where the target measure is defined by the SMM CPU direct measure. The figure 4 shows this relationship.

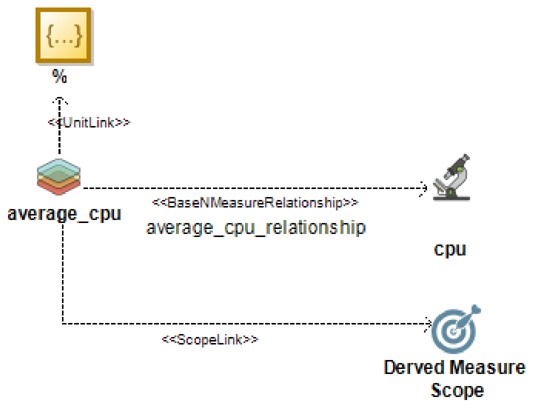


Figure 4: Average CPU Measure Diagram

These SMM collective measures are inserted into the SMM Library of Green Measures and could be referenced by some SMM collective measurements. The latter correspond to the results of statistical analysis mentioned above. They provide information on measurands, for instance the average CPU load, its standard deviation, its minimum and maximum. A measure that stands for the CPU load area would enrich these collective measures in order to provide the CPU load over the time as the same way as the power measure leads to a measure of energy i.e. the area of the power over the time.

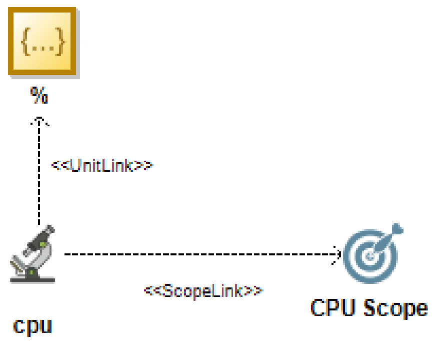
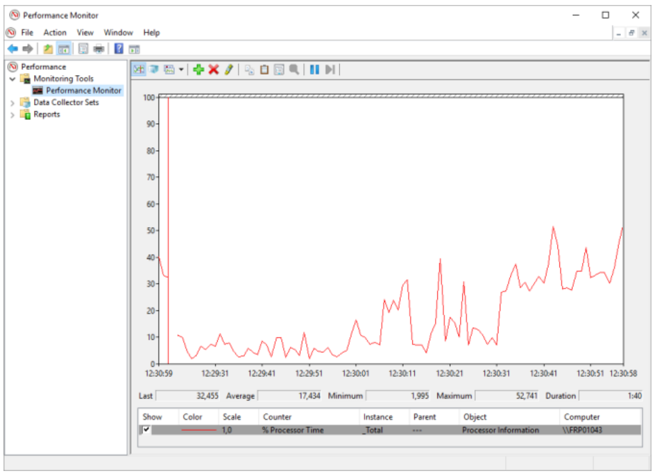


Figure 5: CPU Measure Diagram

Measurements of the CPU percentage can derive from tools like Windows Perfmon.exe as in Figure 6 or from the de MEASURE Energy MonITorig (EMIT for short) as detailed in section V. Measurements for the RAM usage can derive from that tools as well. These tools monitor the computer activities i.e. its processes, their CPU load, their RAM usage, their HDD access, their network bandwidth, etc. However, measurements of the power, the intensity, the voltage or the power factor derive from other kinds of tools as detailed in section V. Such tools monitor the power of computers i.e. their energy consumption.



Last 32,455 Average 17,434 Minimum 1,995 Maximum 52,741 Duration 1:40

Show	Color	Scale	Counter	Instance	Parent	Object	Computer
✓	—	1,0	% Processor Time	_Total	---	Processor Information	\\FRP01043

Figure 6: W10 Perfmon.exe showing average, minimum and maximum CPU %

V. Emit Api

Energy MonITorig (EMIT for short) is a set of tools that aims at monitoring software energy consumption. It is designed as network made of sensors and actuators and can be seen as composed of 5 kinds of tools:

1. The 1st type of tool gathers web-services called "observees". They make possible to launch a process on a computer from a given command line. These tools then allow to launch a software on a computer from a remote computer in order to monitor its implementation. These tools are mandatory for software energy monitoring as they manage software executions.

2. The 2nd type of tool is HTTP services named "power measurement observers". They consists in connected power or energy measurement instruments. A measurement acquisition can be remotely started, remotely stopped and the measurement data can be retrieved once the monitoring is achieved. These tools are mandatory for software energy monitoring as they provide energy measurements. However, their energy footprint does not affect the energy consumption of the monitored software as they do not need to be installed on the computer that hosts the "observees".

3. As for the 3rd type of tool, HTTP services named "computer activity observers", they should be installed on observed computers and then their own energy footprint is monitored by power measurement observers during a measurement

acquisition in which the computer activity observers are involved. The latter consist in programs that scan the operating system logs, directly or not thanks to libraries. As the previous kind of observers, they can be remotely started, remotely stopped and the measurement data can be retrieved once the monitoring is achieved. These tools are not required for software energy monitoring on the first step. However; computer energy consumption is involves by activities such as RAM usage, CPU load, HDD inputs and outputs, network bandwidth, etc (see [12]).

4. The 4th type of tool corresponds to the web-services base SCADA (Supervisory Control And Data Acquisition) application itself. This sort of application embeds HTTP clients that synchronize the different registered “observees” and “observers” for measurement acquisitions. These applications also exposed the collected measurement data throughout HTTP services in order to build third-party tools for reporting or data analysis. These exposed HTTP services correspond to a RESTful API.

5. The 5th and last kind of tool is merely composed of clients of EMIT SCADA applications e.g. graphical user interface applications that enable to control any SCADA systems that complies the previous RESTful API.

EMIT is an extensible set of tools: it comes alongside an API and some design patterns that allow developers to plug their own equipment, devices, instruments, measures, etc.

VI. Emit Experimental Design

EMIT experimental design aims at monitoring software and acquire measurements helpful to compute software energy consumption. The Figure 7 : EMIT Experimental Design describes it. This figure points out 2 kinds of networks: The red network stands for the communication network i.e. here the Ethernet network. The green network stands for the electric network. Whereas the communication network is fully distributed, the electric one is somehow controlled by power measurement instruments. In fact, they are placed between the electricity supplier (the grid) and the monitored computers in order to monitor them. Moreover, they do not have to monitor computers hosting the management system whereas these computers require to be connected to the communication network in order to launch measurements and collect their results.

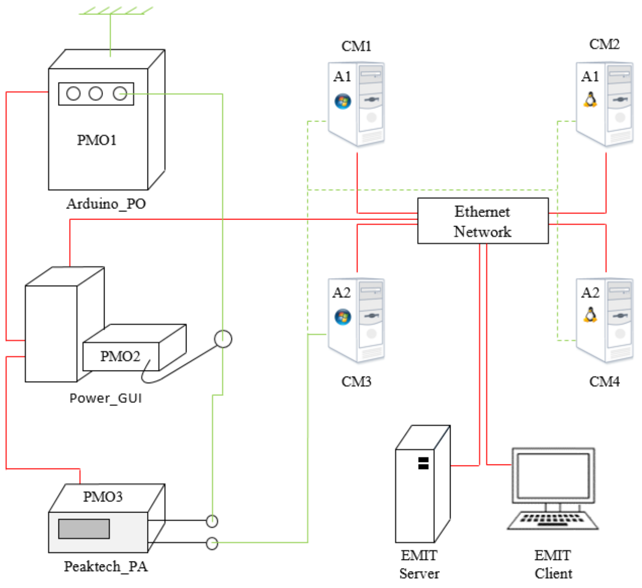


Figure 7 : EMIT Experimental Design

1. “Observees” are located on the four different computers called CM1,…,CM4. Those computers have been chosen because of the combinations between two parameters: the operating system (Linux or Windows) and the architecture (32 bits or 64 bits for example). These computers are connected with EMIT throughout Ethernet.

2. “Power measurement observers” are provided by three devices called Arduino_PO, Power_GUI, Peakteck_PA in the Figure 7 : EMIT Experimental Design. The latter is an industrial power analyser able to monitor one power supply. It outputs data every 300ms throughout a serial port. As for Arduino_PO, it also broadcasts data on a serial port but every 200ms. However, it consists of a 3-way power measurement instrument that is composed of voltage and intensity sensors and controlled by an Arduino microcontroller. These two devices are connected using serial ports to the computer that owns the third observer Power_GUI which is also composed of voltage and intensity sensors but controlled by a National Instruments data acquisition device. All of these power observers can be controlled remotely using web services located on the Power_GUI computer. And they provide time series for the active power.

3. “Computer activity observers” are located on the four computers called CM1,…,CM4. They provide time series for CPU load and RAM memory used currently; however, this can easily be extended in future works. There exists two implementations that are based on different Java libraries: Sigar [13] and

Oshi [14]. The latter are wrapped into web services that start, stop monitoring and provide its data.

4. The EMIT SCADA application is a set of web services that controls all these “observers” and “observees” throughout HTTP requests send to the corresponding observer or observee web services that are hosted on computers CM1,…,CM4 and the Power_GUI one. It requires that exactly one “observee” is enabled at once; it allows none or several observers and manage measurements as the Figure 8 : EMIT Measurement Process shows: Firstly, it starts the “observers”. It then waits for 5 seconds in order to monitor the target computer at idle state. It launches the “observee” i.e. the specified program on the remote computer. After software executions stop, EMIT pauses 5 more seconds before stopping the observers and retrieving their measurement data. It finally stores these data into a database.

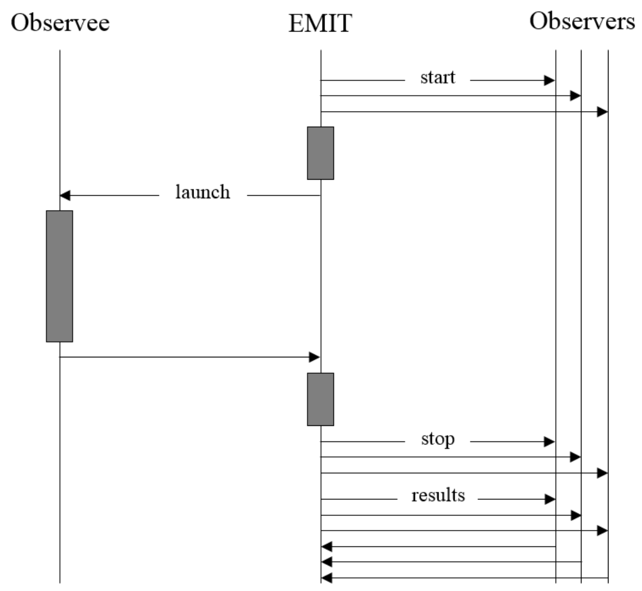


Figure 8 : EMIT Measurement Process

5. These data are exposed to different EMIT clients throughout web services. The latter also allow clients to manage measurands i.e. monitored software and to perform some measurements. Currently, several EMIT command line clients have been developed as well as a standalone Java applications that renders measurement data thanks to the XChart Java library [15].

For example, the Figure 9 : Measurement Data Example shows a chart built on measurement data for a given acquisition. No one analysis are performed by the EMIT SCADA application. It merely consists in collecting data, not analysing them. Analysis can be applied to measurement data by retrieving them from the EMIT server (SCADA application) and by providing the analysis result thanks to web services (see IX).

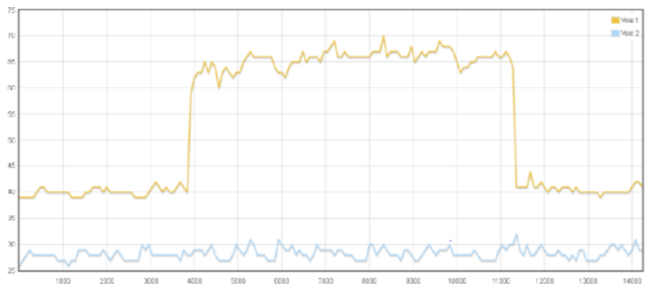


Figure 9 : Measurement Data Example

VII. Data Model

EMIT data model focuses on measurements. In fact, the central entity in the Figure 9 : Measurement Data Example class diagram called *Measurement* references four other entities:

1. The *MeasurementSet* entity that corresponds to the set of measurements collected during one measurement acquisition which is specified firstly by a measurement timestamp thanks to its *measured* attribute and secondly by a measurement *duration* thanks to its so-called attribute,
2. the *Measure* entity i.e. the type of measurement that is defined by its *name* and its *unit* of measure,
3. the *Instrument* entity that corresponds to the observer which provides this measurement that is defined by its URI *identifier*,
4. the *Environment* entity that corresponds to the observee characteristics such as the operating system and the computer architecture. Hence its attribute names *sys*, *arch* and *version.*

The *Measurement* entity is defined by an attribute called *feature* that corresponds to the name of this measurement among all the measurements provided by its observer during a single acquisition. This entity is also defined by an attribute

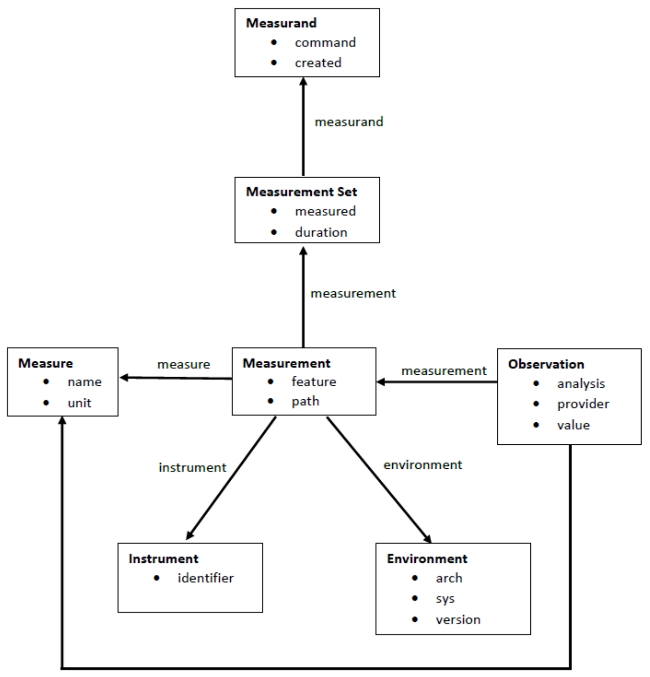


Figure 10 : EMIT Data Model

path which corresponds to the file that contains this measurement data. The latter are formatted as a list of key-value pairs respectively made of long and double values.

EMIT data model is composed of two more entities: *Measurand* and *Observation*. The first one corresponds to the monitored software process specified by its *command* line. The last entity makes possible to register some *analysis* results (*value*) provided by third-party tools (*provider*) over a given measurement. For instance, this could be used for storing some mere statistics such as the average, the An observation is related to a given measure that can be different from that of its corresponding measurement.

VIII. Mapping from Emit to SMM

Structure Metrics Metamodel (SMM, see Structured Metrics Meta-Model (SMM)) has been design for formalizing metrics, measures and measurements for any given domain. EMIT data model as showed in Figure 10 : EMIT Data Model doesn't comply SMM as it has been design to suit the measurement process which is devoted to one measurand and which produces a set of measurements. However, this data model can be embedded into a SMM model as follows:

Every measurements of a given measurement set are sliced according to the different instruments that provide these measurements: each Measurement slice are then mapped into a SMM observation where the *whenObserved* attribute corresponds to the *measured* one of the measurement set and where the *tool* attribute corresponds to the *identifier* of the corresponding instrument. Therefore, each measurement of such measurement slices is mapped into a SMM observed measure.

Every measures related to these measurements are inserted into a unique SMM library as a SMM direct measure. Moreover, every measures related to an observation are inserted into this SMM library as SMM collective measures whose *accumulator* corresponds to the observation *analysis*. Accumulator can range over several statistical functions such as average, minimum, maximum, standardDeviation, etc. Every collective measures own a measure relationship (defined by a BaseNRelationship) between itself and the direct measure of the measurement mapped measure. These collective measures have to be synchronized between the EMIT data model and that of SMM: it is required in order to ensure the mapping compatibility between them. Finally, each EMIT observation is mapped into a SMM collective measurement which share its *value* attribute with the so-called attribute of the EMIT observation. The *measurand* attribute merely corresponds to the *command* attribute of the measurand related to this observation throughout the minimum or the maximum of the measurements data values. measurement and measurement set. The Figure 12 : EMIT Mapping Target Metamodel in SMM illustrates this mapping from EMIT data model into that of SMM. The Figure 11 : SMM Model shows the result of such a mapping rendering with the SMM module in Modelio.

IX. SERVICE ORIENTED ARCHITECTURE

EMIT SCADA application is service-oriented: it provides web services allowing third-party users or tools to manage measurands, instruments and observations i.e. to create, update or delete as well as to retrieve measurements, their measures and their values. Available web services provided by EMIT SCADA application are listed below:

- **measurand/launch**
  This web service accepts POST requests which content corresponds to a measurand formatted as a JSON object, launches a measurement process for this measurand and inserts a measurement set and its underlying measurements generated by the measurement process into the database.

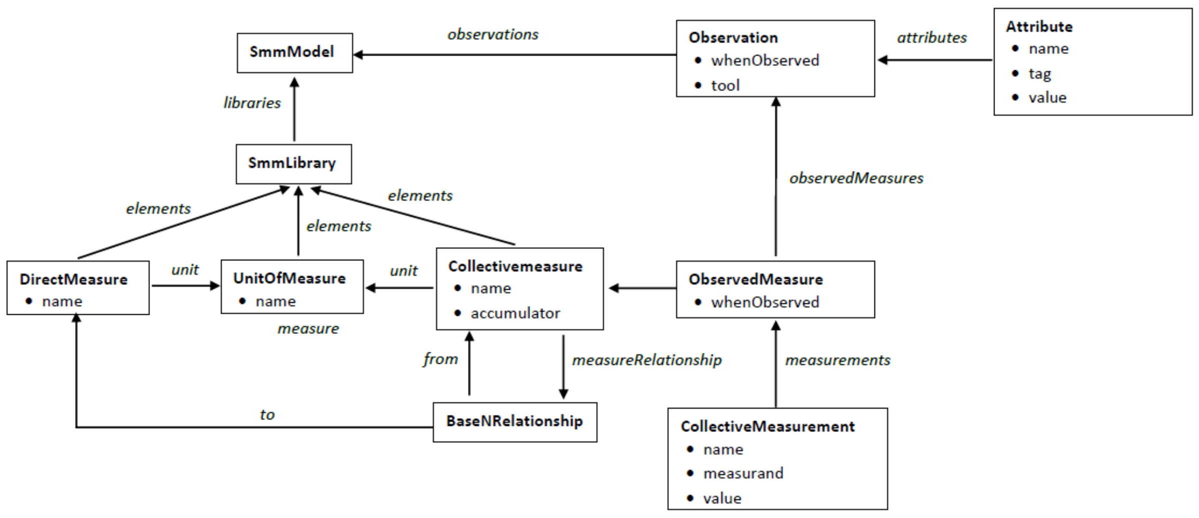


Figure 12 : EMIT Mapping Target Metamodel in SMM

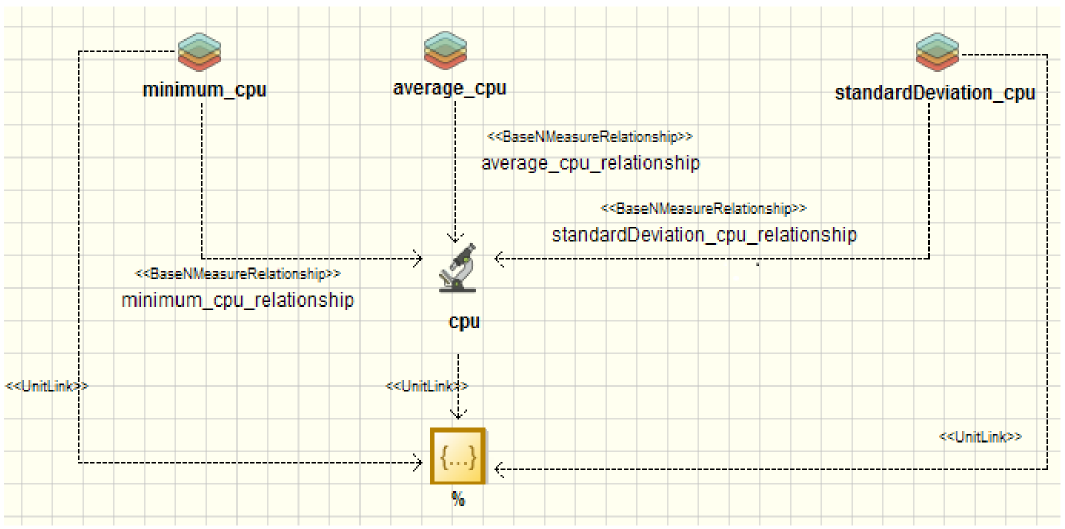


Figure 11 : SMM Model

- **measurand/list**
  This web service accepts GET requests and provides the list of measurands stored in the database in JSON format.
- **measurand/create**
  This web service accepts POST requests which content corresponds to a measurand formatted as a JSON object and inserts this measurand into the database.
- **measurand/update**
  This web service accepts POST requests which content corresponds to a measurand formatted as a JSON object and modifies this measurand in the database.
- **measurand/delete**
  This web service accepts POST requests which content corresponds to a measurand formatted as a JSON object and removes this measurand from the database.
- **instrument/list**
  This web service accepts GET requests and provides the list of instruments (observers, observees) stored in the database in JSON format.
- **instrument/create**
  This web service accepts POST requests which content corresponds to an instrument formatted as a JSON object and inserts this instrument into the database
- **instrument/update**
  This web service accepts POST requests which content corresponds to an instrument formatted as a JSON object and modifies this instrument in the database
- **instrument/delete**
  This web service accepts POST requests which content corresponds to an instrument formatted as a JSON object and removes this instrument from the database.
- **measurementset/list**
  This web service accepts POST requests which content corresponds to a measurand formatted as a JSON object and provides the list of measurements sets related to the given measurand in JSON format.
- **measurement/list**
  This web service accepts POST requests which content corresponds to a measurement set formatted as a JSON object and provides the list of measurements related to the given measurement set in JSON format.
- **measurement/measure**
  This web service accepts POST requests which content corresponds to a measurement formatted as a JSON object and provides its measure in JSON format.
- **measurement/instrument**
  This web service accepts POST requests which content corresponds to a measurement formatted as a JSON object and provides its instrument in JSON format.
- **measurement/environment**
  This web service accepts POST requests which content corresponds to a measurement formatted as a JSON object and provides its environment in JSON format.
- **measurement/data**
  This web service accepts POST requests which content corresponds to a measurement formatted as a JSON object and provides its data in JSON format.
- **observation/list**
  This web service accepts POST requests which content corresponds to a measurement formatted as a JSON object and provides the list of the

observations related to this measurement in JSON format.

- **observation/create**
  This web service accepts POST requests which content corresponds to an observation formatted as a JSON object and inserts this observation into the database.
- **observation/update**
  This web service accepts POST requests which content corresponds to an observation formatted as a JSON object and modifies this observation in the database.
- **observation/delete**
  This web service accepts POST requests which content corresponds to an observation formatted as a JSON object and removes this observation from the database.

As EMIT service-oriented architecture suggests, the single entry point is the first web service *measurand/list* that retrieves the measurands from the database. And the main entity is the *Measurement* one as it is involved as the input of 6 web services.

The set of web services that manage the *Instrument* entity is devoted for controlling the EMIT experimental design configuration.

The set of web services that manage the *Observation* entity is dedicated to the interoperability with analysis algorithms.

X. Hanoï Tower's Use case

Hanoï Tower has been investigated in [8] in order to compare the energy efficiency of several programming languages. Different implementations of the Hanoï Tower recursive algorithm has been compared (including C++, Java, OCaml) and measured using the PowerAPI power measurement instrument. Each implementation has been monitored over some executions with the parameter for the number of disks sets to 15. It led to the conclusion that optimized C++ implementations with the Java one were the ones that consumes the less energy overall.

The experiment has been reproduced by EMIT in order to prove its reliability. Six different implementations of Hanoï Tower has been rewritten as the original ones were not provided alongside [8]. They consists in one Java implementation, 3 C++ implementations (one with none option, the other with the O2 option and the last with the O3 one), one OCaml implementation and one Python one. The experiment was conducted on the same architecture, with the same operating system. 50 measurement processes has been performed by measurands or implementations. Power measurements were achieved by the same instrument: the Power_GUI observer which is the most reliable instrument available on EMIT at that time. Every measurements have been verified according to the algorithm detailed in [5] that makes possible to detect disturbances happening before, during and after measurement processes.

Results on that clean sets of measurements show that the average energy consumption of C++ implementations is the lowest of all implementations as well as [8] presented. In fact, the C++ implementation either with the O2 or with the O3 options has an average energy consumption of 125.4 nano-Joules. In the same way, the OCaml implementation leads to an average energy consumption of 178.4 nano-Joules whereas that of Python has an average energy consumption of 203.6 nano-Joules. These results could match those of [8] as well as they applied a scale on their results with PowerAPI. The surprising result concerns the Java implementation which average energy consumption is measured at 421.5 nano-Joules. Such a result seems in contradiction with [8]. However, the Java implementation has been measured in launching the JVM each time which leads to a constant added energy consumption (see [9]). This convicts us to provide another Java implementation embedded into a web service in order to avoid such an extra consumption.

XI. Reliability of Power Measurement Instruments

EMIT reliability is mainly supported by the reliability of its observers and by its power measurement one. In fact, there is neither scientific nor technical bottleneck over the observee or the SCADA application as the first tool merely launches a command-line process and, as the second one synchronizes observers, observee and store measurement data. EMIT reliability lies in its measurements data provided by its measurement instrument. Moreover, as EMIT aims at being a software energy consumption monitoring tool, its reliability depends that of its power measurement instruments.

Reliability of such instruments (composed of voltage sensor and current sensors) is measured throughout their results i.e. theirs measurements. The reliability of the power measurements is defined by the measurement uncertainty. Monitoring techniques for power measurements are explained in [9]: a standard design consists in a transducer that sends a constant input signal which is monitored by measurement instruments. The uncertainty is then defined as the relative standard deviation over this constant.

For instance, the Arduino_PO uncertainty is ±3.8% which is close to its theoretical error margin. In fact, error margin of its voltage sensor is ± 2% and that of its current sensor ± is 1.5%; this then leads to a sensor error margin of ±3.5%.

Experiments however reveal the influence of temperature on the Arduino_PO reliability. In fact, external parameters that can produce some noise are identified in [10] such as the variation of temperature, that of humidity as well as the vibrations of the instrument itself. This should be taken into account for further investigations on power measurement reliability.

XII. Energy Footprint of Comupter Activity Observers

Another feature required for ensuring some reliable and fine-grained power measurements consists in the energy footprint of computer activity observer. In fact, as such observers are installed on monitored computers, their own activity is monitored by power measurement observers. It is then important to know how much power they require in order to obtain accurate power measurements of measurands.

The assessment protocol is the quite same as that of the power measurement instruments. A constant-like signal is monitored by the means of a computer at idle state i.e. no process is running but those of the operating system. The same power measurement observer monitors all measurement processes in order to provide this computer energy consumption and thus the computer activity observer energy footprint. As 2 computer activity observer are yet provided within EMIT, the first based-on the Sigar Java library [13], the second with the Oshi one [14], 4 use-cases have to be distinguished. The first use-case consists in monitoring the both observers, the second and the third in monitoring only one of thm and the last in monitoring the computer without any of these two observers. 50 measurement processes are performed per use-case.

These experiment results show that measurements without any computer activity observer running yield an average power of 46.1W. This average power sensitively raises to 72.7W with the Oshi-based computer activity observer whereas it raises up to 75.5W both with the Sigar-based observer and with these two observers running. These results point out the huge difference between the computer energy computer consumption with or without an observer. Moreover, the difference between energy consumptions with the first or the second observer is tiny. The surprising results could mean that it will be difficult to draw out software energy consumption from the computer overall energy consumption while some computer activity are running. This requires us to elaborate an analytic method to correlate energy consumption and computer activity involved by a software execution from two disjoints set of measurements i.e. acquired during different measurement processes.

XIII. Conclusion and future work

MEASURE will deliver tools to continuously compare runtime production performance with energy consumption. The results of MEASURE will contribute, during and beyond the duration of the project, to both research and education programs in Computer Science, and specifically the Track in Software Engineering and Green IT. The IT market in Europe will need computer scientists for at least the coming ten years; that of software engineer is the top-1 profession worldwide since numerous years; competencies to measure the quality of complex software will become increasingly crucial for the software industry, and in all top sectors where the role of software is crosscutting and pervasive. Within the MEASURE platform the French cluster will contribute green metrics for ensuring software quality, such definitions will be done using the SMM specification language implemented in MEASURE [6].

Acknowledgment

The research leading to these results was partially funded by the ITEA3 project 14009, MEASURE.

References

[1] MEASURE project web site, http://measure.softeam-rd.eu

[2] MEASURE project description, https://itea3.org/project/measure.html

[3] Structured Metrics Meta-model™ (SMM™) Version 1.1.1. April 2016, http://www.omg.org/spec/SMM/1.1.1.

[4] MEASURE 14009 Measuring Software Engineering, Softeam. ITEA Magazine June 2015, n° 21 pages 25-26, Juin 2015. Available at https://itea3.org/itea-magazine.html

[5] Rocheteau, Jérôme and Gaillard, Virginie and Belhaj, Lamya, How Green Are Java Best Coding Practices, SMARTGREENS 2014 - Proceedings of the 3rd International Conference on Smart Grids and Green IT Systems, Barcelona, Spain pages 235--246

[6] Modelio, https://forge.modelio.org/projects/smm

[7] Structured Metrics Meta-model, http://www.omg.org/spec/SMM/

[8] Noureddine, Adel and Bourdon, Aurélien and Rouvoy, Romain and Seinturier, Lionel , A Preliminary Study of the Impact of Software Engineering on GreenIT - Proceedings of the First International Workshop on Green and Sustainable Software, June 2012, Zurich, Switzerland. pp.21-27, 2012.

[9] An Energy Consumption Model for An Embedded Java Virtual Machines. S. Lafon, 2006.

[10] Rubén, Saborido and Venera, Arnaoudova and Giovanni, Beltrame and Foutse, Khomh and Giuliano, Antoniol, On the Impact of Sampling Frequency on Software Energy Measurements, PeerJ PrePrints, Tech. Rep., 2015.

[11] J. Rocheteau, "Energy Wasting Rate as a Metrics for Green Computing and Static Analysis," in MegSuS 2015 - Proceedings of 2nd International Workshop on Measurement and Metrics for Green and Sustainable Software, Cracow, Poland, N. Condori-Fernandez, G. Procaccianti, C. Calero, and A. Bagnato, Eds., oct 2015.

[12] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," ACM SIGOPS Operating Systems Review,vol. 47, no. 3, pp. 42–49, 2013.

[13] Sigar library, https://support.hyperic.com/display/SIGAR/Home.

[14] Oshi library, https://github.com/dblock/oshi.

[15] XChart library, http://knowm.org/open-source/xchart.