Representation and Visualization of Data for Software Energy Efficiency

Remco M. Havermans

Computer Science

VU University Amsterdam

A thesis submitted for the degree of

Master of Science

Computer Science - Software Engineering and Green IT

2015
1. First Reader: Giuseppe Procaccianti (VU University Amsterdam)

2. Second Reader: Patricia Lago (VU University Amsterdam)
Abstract

ICT has become a major element in everyday life, it presents a significant impact on the global energy consumption. Hardware has typically been pointed out as the area for improving the energy efficiency of ICT, even though it has become increasingly more efficient. The role of software in the consumption of energy has for the most part been neglected.

Research has proven that software has a significant role in the energy consumption of ICT, however the research of understanding the exact role is still at an early state. With the increasing demand in power, understanding and reducing the energy consumption of software has become a top priority for upcoming research.

The objective of this thesis is to support the understanding of the relationship between software and energy consumption for analysing energy efficiency. The problem is addressed in two parts: first it investigates how software and energy can be represented in a data model, secondly it looks into visualization methods for analysing and understanding the data and the relation between software and energy.

Using knowledge extracted from literature, we present the Software Energy Consumption Model, a semantic model that can represent data relating to software and energy. The model contains the basic relationships between software and energy, and due to its semantic nature future analysis of the data and the model can uncover new relationships.

Furthermore, the second part of the thesis looks into two visualization methods for understanding the relation between software and energy. First of all, we show the usability of dataset analysis for uncovering relationships between software and energy using the SynerScope tool. Secondly, we demon-
strate ‘Energy’ flame graphs, a novel adaptation of ‘flame graphs’ that combine runtime information of software and the consumption of energy.

Both the model and the visualization methods are validated using experiments, demonstrating their usability in relation to software consumption. The presented methods can support future research for understanding the complex relationship between software and energy, and improve the energy efficiency of software.
# Contents

List of Figures vi

List of Tables vii

1 Introduction 1
   1.1 Energy in ICT ........................................ 1
   1.2 Software Energy Consumption .......................... 2
   1.3 Research Questions ................................... 2
   1.4 Research Method ...................................... 4
   1.5 Structure ............................................ 4

I Modelling software and consumption 7

2 Background 9
   2.1 Introduction ........................................... 9
      2.1.1 Context ........................................... 9
      2.1.2 Objective ......................................... 10
      2.1.3 Research Question ................................. 11
      2.1.4 Research Method .................................. 11
   2.2 Model Cornerstone .................................... 12
      2.2.1 Model Goal ........................................ 12
      2.2.2 Granularity and Scope ............................. 12
      2.2.3 Requirements ..................................... 13
   2.3 Format Selection ...................................... 15
      2.3.1 Web Ontology Language ............................ 15
2.3.2 Existing Models ........................................ 15
  2.3.2.1 Search Strategy .................................... 16
  2.3.2.2 Inclusion Criteria .................................. 16
  2.3.2.3 Environment Models ................................ 17
  2.3.2.4 Software Specification Models ..................... 18
  2.3.2.5 Software Measurement Models ..................... 20

3 The Software Energy Consumption Model 21
  3.1 Ontology Model ........................................... 21
  3.2 Entity selection ......................................... 21
    3.2.1 Environment ........................................ 24
    3.2.2 Execution Unit ...................................... 24
    3.2.3 Execution Workload ................................ 24
    3.2.4 Function ............................................ 24
    3.2.5 Hardware Environment ............................... 25
    3.2.6 Hardware Resource ................................. 25
    3.2.7 Measure .............................................. 27
    3.2.8 Measurement ........................................ 27
    3.2.9 Module ................................................ 28
    3.2.10 Operating System .................................. 29
    3.2.11 Program Construct ................................. 30
    3.2.12 Program Library ................................... 31
    3.2.13 Programming Language ............................. 31
    3.2.14 Resource ............................................ 31
    3.2.15 Software Application ............................... 32
    3.2.16 Software Environment ............................. 32
    3.2.17 Software Resource ................................. 33
  3.3 Model connection ......................................... 34
    3.3.1 Open Multi-Net ..................................... 34
    3.3.2 Linked Source and Documentation Ontologies ...... 35
    3.3.3 KOontR .............................................. 35
    3.3.4 EvoOnt .............................................. 35
    3.3.5 TwoUse .............................................. 36
# CONTENTS

3.3.6 Software Measurement Ontology ........................................... 36

4 Model Case Study ................................................................. 39
  4.1 Evaluation Case ............................................................... 39
  4.2 Data Identification and fitting ............................................. 39
  4.3 Discussion ........................................................................... 44
  4.4 Threats to validity .............................................................. 45

II Data Visualization ................................................................. 47

5 Visualization Background ......................................................... 49
  5.1 Introduction ......................................................................... 49
  5.2 Context ................................................................................ 49
  5.3 Objective ............................................................................. 50
  5.4 Existing studies ................................................................. 50

6 Analysing Software Energy Efficiency Data .................................. 53
  6.1 Introduction ......................................................................... 53
  6.2 Preparation .......................................................................... 54
  6.3 Experiment .......................................................................... 54
  6.4 Discussion ............................................................................ 58

7 Graphical Representation of Software and Energy ......................... 61
  7.1 Introduction ......................................................................... 61
  7.2 “Energy” Flame Graphs ......................................................... 62
  7.3 Proof of Concept .................................................................. 63
    7.3.1 Data Input ..................................................................... 63
    7.3.2 Flame Graph Visualization .......................................... 65
    7.3.3 Differential Energy Flame Graphs .................................. 67
  7.4 Experimental Setup ............................................................. 69
  7.5 Discussion .......................................................................... 70
  7.6 Threats to Validity .............................................................. 74
## CONTENTS

### III Conclusion

#### 8 Conclusions

8.1 Contribution ......................................................... 77

8.1.1 RQ 1: "What is a suitable data model to represent software and energy consumption?" ............................................. 77

8.1.2 RQ 2: "How to visualize the most relevant entities and relationships of the data model for software energy efficiency?" ..... 78

8.1.3 The Main Research Question ................................. 79

8.2 Future Work .......................................................... 80

### A Data Model

#### References

References
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Model Representation</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Fitted hardware environment</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Fitted memory module</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Fitted software environment</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Fitted software execution</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>Fitted measure</td>
<td>43</td>
</tr>
<tr>
<td>6.1</td>
<td>Energy Usage MySQL</td>
<td>55</td>
</tr>
<tr>
<td>6.2</td>
<td>Energy Spikes Identification</td>
<td>56</td>
</tr>
<tr>
<td>6.3</td>
<td>Memory Usage Pattern</td>
<td>56</td>
</tr>
<tr>
<td>6.4</td>
<td>Memory Usage Comparison</td>
<td>57</td>
</tr>
<tr>
<td>6.5</td>
<td>Apache Watts and Memory</td>
<td>58</td>
</tr>
<tr>
<td>7.1</td>
<td>Flame Graph</td>
<td>62</td>
</tr>
<tr>
<td>7.2</td>
<td>Stacktrace merging</td>
<td>65</td>
</tr>
<tr>
<td>7.3</td>
<td>Flame Graph Visualization</td>
<td>66</td>
</tr>
<tr>
<td>7.4</td>
<td>Flame Graph Visualization</td>
<td>68</td>
</tr>
<tr>
<td>7.5</td>
<td>Stacktrace time-frame selection</td>
<td>69</td>
</tr>
<tr>
<td>7.6</td>
<td>Quicksort Processing Overview</td>
<td>71</td>
</tr>
<tr>
<td>7.7</td>
<td>Quicksort Process</td>
<td>71</td>
</tr>
<tr>
<td>7.8</td>
<td>Differential Energy Flame Graph</td>
<td>72</td>
</tr>
<tr>
<td>7.9</td>
<td>Function Energy and Time Consumption</td>
<td>73</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Overview of all generalized measures identified in the literature review \[1]\[10]
2.2 List of the requirements that are identified for the construction of the model. \[14\]
1

Introduction

1.1 Energy in ICT

ICT has become an important factor in the world’s energy consumption. A study in 2006 showed that 2 percent of the global CO₂ emissions can be attributed to ICT [2], including the emission by the usage of ICT components and its infrastructure. The study notes that the industry need to start “gain a better understanding of the full life cycle of ICT products and services, and innovate to reduce environmental impact.” [2]

Studies by Greenpeace indicate that the rapid growth in usage of internet and cloud services has increased the demand in power, ranking in the top 10 alongside countries [3]. Estimated is that, till 2020, data centres will grow by 9 percent with growing the carbon emissions by 7 percent each year [4]. Therefore, it’s not surprising that energy efficiency is considered a critical element for data centre operations [3].

One of the strategies proposed in by Gartner et al. [2] is to “start measuring power consumption.” A process which usually focusses on hardware components. However, as noted by Eder et al. [5] the energy consumed by computations on the hardware is, in the end, controlled by the software. An experimental evaluation by Procaccianti shows that, for instance, optimizing a data retrieval instruction (query) can reduce the energy consumption as opposed to an non-optimized counterpart [6].

Tools that are able to provide developers with information about the efficiency of their code is envisioned by Eder et al. [5]. The participants of GREENS, a workshop
focusing on efficient and sustainable software, share similar conclusions stating that there is need for research to identifying, and understand, wasteful program code \[7\].

\section*{1.2 Software Energy Consumption}

For improved energy efficiency in IT hardware is often the main target for improvement. As shown by Koomey et al. \[8\] the computations, i.e. the number of operations, per kilowatt-hour has doubled every 1.57 years since 1946, proving a significant improvement of the energy efficiency of hardware.

However, the hardware is eventually controlled by software which decides what instructions have to be performed. As mentioned by Procaccianti \[6\] modern hardware devices are not proportional regarding the load (the utilization of the component) and energy consumption, which makes software an important component for improving the energy efficiency in ICT.

In previous research \[1\], I analysed the measurements used in different techniques to model, investigate, or analyse the energy consumption of software in certain \textit{execution contexts}. The article defines \textit{execution context} as “[..] the environment in which a program is executed.” \[1\].

The article concludes on the lack of cohesion between the found methods. Not only between the types of data used by each of the methods, but also in terms of the specific ‘execution context’ the methods applies to. The application of any of these methods will require a specific dataset and a clear understanding of the context it targets. A generalized model can account for these differences and combine the estimation into a single result.

Another important aspect for improving the energy efficiency of software is understanding the data. Visual representations of the data can provide a significant contribution for discovering patterns, or abnormal behaviour, within software.

\section*{1.3 Research Questions}

Hardware has seen significant improvement in energy efficiency, and is typically seen as the starting point for reducing the energy consumption of ICT. Research has proven
that software has a role in the actual consumption of energy [6], however the definition of this role is still at an early stage.

As software has a role in the energy consumption, there is the question of efficiency: which piece of software can provide it services more efficiently in terms of energy? The exact definition of energy efficiency is complex as it compares two or more items together which have a shared context, i.e. if two unrelated items are compared, a statement of their efficiency is meaningless. However, before the energy efficiency can be discussed, the role of software in the consumption of energy must be known.

In order to understand how software influences the energy consumption two things are required, first the collection of data related to the energy consumption and secondly visualization methods to provide understanding of the data to interested parties.

For this we conceive a semantic model that can be used to store the relevant information and understand the relationships within the data, allowing for the discovery of patterns and abnormalities of software energy consumption. Furthermore, to provide better means of understanding the data, we look into methods for visualizing the collected information.

In general our goal is to provide a representation for the data, as well as exploring possibilities for visualizing said data. The main research question is stated as:

RQ. “How can data relating to software and the consumption of energy be represented and visualized for analysing energy efficiency?”

The research question can be split into specific questions, each addressing a part of the main research question.

• The first part will investigate the data model that can be used for storing data related to the energy consumption of software. This model is required to allow the correlation between different experimental datasets and enriches the available data by adding contextual information for analysis.

RQ 1. “What is a suitable data model to represent software and energy consumption?”
1. INTRODUCTION

- The second part focuses on methods that can be used to visualize the relevant information for showing the efficiency of software. Visualization helps to improve the understanding of large amounts of data and help developers or users understand the energy consumption of software.

*RQ 2. “How to visualize the most relevant entities and relationships of the data model for software energy efficiency?”*

1.4 Research Method

The role of software in the energy consumption of ICT has become an active topic. Although many studies are already devoted to research this relationship, we expect the need for future research in this topic.

This thesis explores different methods that can be used to support future research, enabling the sharing and analysis of data related to software energy efficiency. The following techniques are used in this thesis:

- *Literature Review.* This method provides the means to structurally find background knowledge on the subject, while providing freedom in the selection of sources. Using this method, we gather existing data models for topics relating to the energy consumption of software.

- *Proof of Concept.* This method is used to define the feasibility and usability of an idea. The concept is done together with an experiment, showing the potential of the idea. We use this method to investigate new methods for analysing data for software energy efficiency.

1.5 Structure

The thesis is structured in two parts, the first part will discuss a generic data model for the storing of data related to software energy efficiency. Chapter 2 investigates into the requirements for a generic model and studies literature into the subject. The definition of this model is presented in 3 followed by an evaluation and discussion of the resulting model in chapter 4.
In the second part, this thesis investigates novel techniques for visualizing data relating to software energy consumption. Chapter 5 establishes the background knowledge and presents commonly used techniques. Two methods for analysing data are discussed in chapters 6 and 7, regarding the usage of a program or a special type of graphical representation.

The report finishes with the concluding remarks and future research on discussed subjects in chapter 8.
1. INTRODUCTION
Part I

Modelling software and consumption
2

Background

2.1 Introduction

Various methods for analysing or modelling the energy consumption of software exist. These methods often use different models, data, and formats. A model that can combine all the related data for these methods can be valuable resource for the modelling of energy consumption, and provide options for reuse of data.

The following section describes the required properties of such a model. As well as a short search towards existing models that can be (re)-used.

2.1.1 Context

The impact of software on the energy consumption is an active topic. Although many papers already address the issue of measuring the consumption of software itself, there is a lack of agreement between studies. Different methods and measurements are used to model and estimate the energy consumption. The usage of the measurements can also difference between studies, one study could use the active time spend processing the application, while the other only uses the total time required.

Studies address the energy consumption in a different way, creating methods that use other data to collect the efficiency or consumption of an application. As an example, some studies focus on the time spend processing an application while others investigate the what instructions are actually being processed. As such each new method require a different experimental setup to collect the data the particular method requires. Which,
2. BACKGROUND

in turn, results in each study presenting their own dataset without room to compare the results of different sets with other methods.

Furthermore, the relation between software and the energy consumption is complex and contains many variables. Representing these problems is not a plain task and are often forgotten, even though they have a significant value to the data.

A generic model can provide a standardized format to store the data collected by future experiments, promoting the reuse of the experimental data as well as defined as established relationships within the data.

2.1.2 Objective

The focus of the study is to find or define a model that is able to represent collected data from the analysis of a software application in the environment it was executed. The model should provide a generalized basis for different analysing methods and the measure they require.

A preliminary literature study \[\text{[1]}\] was executed in order to find the measures that were used by different methods for modelling the energy consumption of software. The study collected the results from 71 research papers and identified 31 (generalized) measures that were used, which can be found in table \[2.1\].

These measures form the basis of the different types of measurements the model should be able to represent.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Measure</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time</td>
<td>Frequency</td>
<td>Code Size</td>
</tr>
<tr>
<td>Energy Readings</td>
<td>Cache Misses</td>
<td>Execution Path</td>
</tr>
<tr>
<td>Workload</td>
<td>Operations</td>
<td>Architecture</td>
</tr>
<tr>
<td>Memory Footprint</td>
<td>Libraries Used</td>
<td>Current</td>
</tr>
<tr>
<td>Instructions</td>
<td>Threads</td>
<td>Network Latency</td>
</tr>
<tr>
<td>CPU Cycles</td>
<td>Screen Properties</td>
<td>Datatypes</td>
</tr>
<tr>
<td>CPU Utilization</td>
<td>Network Bandwidth</td>
<td>Language Constructs</td>
</tr>
<tr>
<td>Component Usage</td>
<td>Disk Workload</td>
<td>Number of Components</td>
</tr>
<tr>
<td>Voltage</td>
<td>Energy Delay Product</td>
<td>System Functions</td>
</tr>
<tr>
<td>Hardware Information</td>
<td>Throughput</td>
<td>Network Package Size</td>
</tr>
<tr>
<td>Service Rate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.1:** Overview of all generalized measures identified in the literature review \[1\].

2.1 Introduction

2.1.3 Research Question

The study is focused on finding a model that can be used to define data that is collected while analysing the energy consumption of software. The underlying research question, established in section 1.3, is defined as:

“What is a suitable data model to represent software and energy consumption?”

The keywords from the research question are further defined as follows:

A data model is considered as any type of model that can provide meta information, such as definitions and relations, on the data it represents.

Suitable is defined by the Oxford Dictionary as “right or appropriate for a particular purpose or occasion.” Following this definition in the context of the research question this study defines suitable as “appropriate for representing information on software applications and any data required to analyse its energy consumption.” With an extra emphasis on any, meaning that it should provide room for new types of data to be added in the future.

2.1.4 Research Method

The research is conducted through a structured approach, however no formal guidelines are used.

First the model definition is described, containing the considerations, such as the goal and requirements, for the de final model. Furthermore, a short literature search is executed to find existing models that can either fit the requirements of the model, or provide reusable components for the model.

Following the definition, the model itself is described and explained. The components that exist within the model are addressed as well as the usage of said components. Any links towards the other found models are explained here.

The resulting model is evaluated against an existing use-case. The data provided from the use case is applied to the defined model and the outcomes of the model are discussed.
2. BACKGROUND

2.2 Model Cornerstone

The cornerstone of the model provides the fundamental principles, such as the goal and requirements of the construction of the model. The basis for the model is formed using these principles and can be used for the evaluation of the resulting model.

2.2.1 Model Goal

The goal of the model, as defined in the objective, is to provide a generalized data model that can be used by different methods for purposes related to the energy consumption of software applications.

There are two main purposes for the model, the first is to promote the storing and sharing of experiments. Allowing the reuse of existing data to improve the analysis of software in relation to energy consumption.

A second goal is the discovery of patterns within the data. Although the model should already contain the (basic) relationships between the data, structuring the data in a (formal) semantic model allows the discovery for correlations or relations within the dataset that would not emerge otherwise.

2.2.2 Granularity and Scope

The goal of the model is important for setting the granularity required in the model. A ‘thing’ within the model consists of two parts, the description (concept) and the specific instance of ‘a thing’ (entity).

These definitions can be different depending on the scope, for example model A is only interested in objects and defines ‘vehicle’ as a description something a person can own, with a ‘car’ being a particular instance of the ‘vehicle’ listing it as “Car of Brand X.” Model B on the other hand, wants to know about the particular properties of a specific car. B states ‘car’ as the description, whereas the actual car is defined as the entity such as “Car X with licence 123 owned by Y.”

The model of this study will restrict itself to the components needed to provide support for the measurements shown in table 2.1. These identified measures origin from existing methods for the calculation or estimation of energy efficiency. As a result the classes within the model provide a basis that can be used by various existing approaches, and provide an instant validation into the usability of the model.
2.2 Model Cornerstone

Even though the model is restricted to the identified measures, it strictly provides the basis required to include these results. Therefore the open world assumption, which states that missing parts are irrespective to whether they validly exist, is an important aspect of the final model. For instance, reusing the previous example, if model B does not describe that a ‘car’ can have a ‘hitch’ it will not mean that there is no ‘car’ with a ‘hitch,’ it simply states that the connection is yet to be made. The assumption allows for future extensions to incorporate more or specific data, which at the moment of creation were not yet defined.

The model will need to incorporate different notions of ideas, such as the description of the computer used in an experiment as well as the structure of a piece of software. In order the level of detail for the descriptions, the model is limited to an overview of the software structure.

The actual items used within a part of the structure, the ‘source code’, are influenced by a large set of parameters and requires extensive research for an accurate definition of its relations. Although some models use these concepts to perform specific estimations or calculations, they are left out of scope of the model. This exclusion is justified as the model is designed for form the basis to containing data related to the energy consumption of software. A separate (sub) model can provide specific concepts for these aspects, for the methods that require them.

2.2.3 Requirements

Even though some requirements of the model have already been defined, a formal specification of the requirements provides the guidelines for reasoning for the selection or inclusion of a model. The requirements can, after the model definition, be used for the evaluation to discuss if they have been met.

All requirements that define the guidelines for a suitable data model to answer the research question are defined in table 2.2 with each an unique identifier, definition, and rationale.

The definition of these requirements is based on the defined goal, and literature reviewed in the preliminary study [1] alongside its concluding remarks. The first three requirements define the relevant information of the relationship between software and energy: the used environment (RQ1) and the gathered observations (RQ3) which are grouped together by specific execution (RQ2) that defines the work performed.
2. BACKGROUND

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>The target model needs to have a clear definition of the environment a program is executed in.</td>
<td>The environment can be influential to the behaviour of a program and therefore needs to be included within the model.</td>
</tr>
<tr>
<td>RQ2</td>
<td>The target model should provide room for including multiple executions and distinguish the results based on each execution.</td>
<td>Typically an experiment is executed multiple times in order to compensate for unknown variables. It is important that the collection of these results can still be mapped to the execution that produced them.</td>
</tr>
<tr>
<td>RQ3</td>
<td>The target model must be able to store dynamic data that is linked to the point in time it was observed.</td>
<td>An important aspect of the model is the incorporation of energy consumption data, it therefore must provide the means to include such data and identify it based on the moment in time it was observed.</td>
</tr>
<tr>
<td>RQ4</td>
<td>The target model needs to make room for extension or further definition of the basic model.</td>
<td>As the model is defined to be non-absolute and extensible, the format used to define the model must provide support to further extend the target model.</td>
</tr>
<tr>
<td>RQ5</td>
<td>The target model must be able to include all the measurements identified in table 2.1, excluding specific items focusing on the ‘code-base’ of an application.</td>
<td>The purpose of the model is to include the data taken from methods for the analysis of software energy consumption, and thus should be able to define them within the model.</td>
</tr>
</tbody>
</table>

Table 2.2: List of the requirements that are identified for the construction of the model.

The preliminary study [1] provided a list of measures used in various methods for measuring energy consumption of software, these measures provide a basis of data which influence the energy consumption. Therefore, the inclusion of these measures would provide a basis for defining the relationship (RQ5). The exact used measures differentiate between methods, illustrating the complexity of this relationship. In order to account for future understanding of software energy consumption, the model needs to be flexible in terms of extension (RQ4). When the role between software and energy becomes further defined, the flexibility allows for the improvement on the model without invalidating existing data.
2.3 Format Selection

The format of the model defines the basic possibilities of the model. Although data can be placed in tabular formats, the links between tables provides significant information of the data itself. Such semantic layers can provide strong additional information that could increase the data value.

2.3.1 Web Ontology Language

The Web Ontology Language (OWL) is a language that allows for modelling knowledge with a semantic layer. Reasoners can use the semantics to validate the models and derive logical consequences from the links between its concepts. Due to the underlying XML schema construction, the model has the ability to link multiple models together.

There are several advantages of using OWL for modelling green measurements in a model. The reasoners can be used to find implicit connections between experimental data, for instance it could be possible to identify systematic issues (hotspots) and implicit properties of the part of the program where it was found, which in turn could lead to the discovery of energy expensive issues in the underlying programming language, or identifying functions consuming a lot of resources.

An important property OWL provides is the option to link and combine different ontology models together. Not only can other ontologies be used to form a strong basis for the model, additional models can be created to further specify the data placed into the model. For instance, a future research could attribute programs with their formal specification of requirements for analysis, suggestions for improvements, and possibly even validation.

These properties make OWL an excellent choice for the purpose of the definition of software energy consumption knowledge. OWL can provide the connection between the concepts that are required for the consumption data and provide a flexible model that can easily be extended.

2.3.2 Existing Models

Various research has already been conducted defining ontology models for different subjects and domains. The OWL Web Ontology Guide states that “in order for ontologies
to have the maximum impact, they need to be widely shared.”\[9\] Therefore, an ex-
ploratory search is executed for other ontologies that can provide a strong basis for
the model. Using this basis the defined ontology can share concepts between models,
improving the impact of both ontologies.

The search strategy is defined first, with the criteria before the different models
found are discussed.

2.3.2.1 Search Strategy

The search for other ontology models is not performed in a systematic way. Websites
that index different ontologies are rare, often not maintained, or missing the meta-data
required to perform a structured and systematic search. The review is restricted on
ontologies that are described in published papers. This allows for a semi-systematic
review to be performed on ontologies that are peer-reviewed. The inclusion criteria will
go more into detail on the definition for the \textit{quality} aspect of ontology models.

The focus is placed on the potential overlap with other ontology models containing
more specifications for the said domain. Therefore, the search is split into three different
parts: identification for ontologies that specify the environment (software or hardware),
ontologies that provide specifications for software applications, and a formal definition
for defining different measurements from a measured application.

Each part defines separate keywords for a search query customized for their re-
spective parts. The search engine used is Google Scholar, which also links to other
databases such as IEEE Xplore and ACM Digital Library. Selected papers are reviewed
and compared to the inclusion criteria.

2.3.2.2 Inclusion Criteria

The reuse and connection of other models is an important benefit of the language. Welty
et.al. notes in The Web Ontology Language Guide that: “if you can find an existing
ontology that has already undergone extensive use and refinement, it makes sense to
adopt it.”\[9\] However, if there is no value gained from the inclusion of another model,
there is no benefit for adapting the models. The inclusion of other models without any
purpose can even pose constraints on the conceived model, and threaten the flexibility
it requires.
2.3 Format Selection

The selected models are reviewed using a predefined set of criteria. This list of criteria are used to determine when a model is considered usable and valuable for inclusion in the final model. The following criteria, with their rationale, are used to assess if the found ontologies are suitable for inclusion:

1. The selected ontology must have a published version of the model definition. The definition of an ontology model is typically improved over a long period of time. In order to make an accurate assessment of the model, it must have published definition of the model. A specific version of an ontology model cannot change its definition, thus remains a valid inclusion in the future.

2. A selected ontology must have the goal and purpose clearly defined. Knowledge is typically ambiguous and tied to a specific domain. This poses the risk of misinterpretation if the context of the model is not specified. Mismatched concepts present a serious threat against the validity of the resulting model.

3. The selected ontology must contain a specific goal and not provide a generic layer for other ontologies (also known as a Upper- or Foundation Ontology). Various ontologies exist that provide generic concepts that can be reused in different ontologies. Linking said ontologies can be useful for specific purposes, however hold no added value in the current context of the study.

2.3.2.3 Environment Models

The environment of executing software is an important aspect of the consumption model. Hardware or software components can directly influence how code is executed in a certain environment due to differences amongst similar components. When the data of the energy usage from software is compared, the context in which it was executed can have an impact on the gathered results, therefore they should be taken in consideration by methods calculating or predicting the energy consumption.

The search query is built up taking the keywords for the target models (hardware or software environment), in combination with similar keywords resulting in the following query: ontology “(hardware OR software) (infrastructure OR environment OR architecture).” Because the study only analyses the initial results are analysed, the search string was executed twice using either hardware or software in contrast using a combination of
2. BACKGROUND

both, hardware OR software, as specified in the query. The top 30 results of both query executions were taken and combined. Creating an equal set of articles that dedicate to both the hardware and software environments, resulting in a total of 60.

Open Multi-Net

The Open Multi-Net proposed by Willner et al. [10] is a semantic approach for describing enterprise hardware infrastructures. It defines the components that exist in such infrastructures as well as the relationships between them.

The ontology is divided into an upper ontology with various sub-ontologies focusing on a specific part of the entire infrastructure. The component sub-ontology is a potential location that can provide useful additions, as it defined the components used within a machine (such as CPU and memory).

Infrastructures that are already defined can be included within the defined model, providing information about the environment where an experiment has be performed. This additional set of information can uncover previously unknown correlations between the components and the execution of a software application.

2.3.2.4 Software Specification Models

The specifications of software are an important aspect to any analysis of the software. Many programming languages exist that can be used to create applications. Depending on the language there are libraries defined that can provide proven and tested methods that aid the software development process. As such it is important that the specifications of the software are known.

The specifications are not restricted to high level concepts such as the language and used libraries. As some libraries can be large, only a small set of functions might actually be used by the application or even the (misuse) of certain functions might effect the energy performance of an application.

The search towards specification ontologies was done using the search query: ontology AND “(software OR application) (specification OR definition OR modelling)” of which the 50 most relevant results (according to the search engine) are taken. The authors Zhao et al. [11] present an ontology classification for software engineering, the ontologies mentioned in the article for the purpose of software specification where added to the search results.
Linking Source and Documentation Ontologies

The model proposed by Zhang et al. [12] focusses on the representation of existing code. The authors proposed a source code and documentation ontology that is linked together using a Text Mining system to provide traceability in the code. Although the focus is on software maintenance, it can provide a valuable asset to the consumption model to show, and infer relations between artifacts.

Further work on this approach is done by Witte et al. [13], suggesting an approach for automatic population of the ontologies by parsing the source code (Java in the example) and extract the information.

KOontR

Happel et al. propose the solution KOontR [14] focusing on software reuse. Their approach suggests an architecture that uses ontologies to store the application knowledge. Using SPARQL they query back results on specific properties, such as checking if an artifact can be used under a proprietary license.

Although the ontology used in the prototype is minimal, it suggests inclusion of domain ontologies to further specify applications. Systems using this approach could provide extra information on the application that is tested upon, and the energy data can also attribute by allowing software reuse for artifact that are proven to be more energy efficient.

EvoOnt

The ontology EvoOnt proposed by Kiefer et al. [15] is designed to exchange software information. The ontology is divided into three sub-ontologies: the software ontology model (SOM), bug tracking ontology model (BOM), and a version ontology (VOM).

The SOM sub-ontology is used to represent the constructions used in object oriented code, it additionally contains entities for modelling measurements (defined as metrics in the ontology). The BOM and VOM ontologies are used to define bug and version information respectively.

In the article, Kiefer et al. use the ontology in combination with their iSPARQL framework, based on the query language SPARQL, to show the ability to analyse the evolution of software over time.
2. BACKGROUND

TwoUse
An interesting suggestion is presented by Parreiras et al. [16], where the authors suggest a combination with the Unified Modelling Language (UML). The TwoUse approach suggests a new formal syntax that can be transformed into the OWL or code syntax. The paper presents the approach to aid in the development process and discusses how it can be applied for maintenance and reuse.

As the syntax also translates to UML, the internal structure of a program is known and modelled. This can be used for the identification of the application when analysed with energy consumption data.

2.3.2.5 Software Measurement Models

Software measurement is an active field in research, using data collected from applications in order to infer or predict information. As software measurements are an intensively researched field, ontologies constructed can contribute by providing a basis for the modelling of measurement data within the model.

The ontology search was conducted using the query: software AND measurement AND ontology. As the study is preliminary only the 50 most relevant results (according to the search engine) were listed and analysed for the criteria. The articles provides a single general ontology for software measurement.

Software Measurement Ontology (SMO)
The initial proposal of the model is defined in [17]. The model serves a double purpose, first to create consensus on the terms used for measurements, and second a common conceptual model for software measurements. The terms used were taken from various ISO standards used in the field of software measurement.

García et al. continues to build on the model in [18] and formalizes the new model as the ‘Software Measurement Ontology’ (SMO). The paper further refines the model using ISO standards to create consensus for different fields and provide a general vocabulary.

The model is implemented in two different management infrastructures for project measurements. The infrastructure proposed by Kunz et al. [19] uses the initial model proposal and extends the model with object-oriented measurements to define the ‘Object-Oriented Metrics Ontology.’ García et al. [20] include SMO in the definition of a framework for measurement of entities in the software lifecycle.
The Software Energy Consumption Model

3.1 Ontology Model

The Software Energy Consumption Ontology model is defined as a self-sustaining model, meaning that its own interpretation of the concepts are defined within the model and any overlaps or connections with other models are not part of the core ontology. However the ontologies that have been identified in the previous chapter (section 2.3.2) are used as inspiration for the Software Consumption Ontology and reused to improve the connectivity with other ontologies.

A graphical representation of the model is shown in figure 3.1 using an UML class diagram. The entities are defined as classes, object properties as association and attributes as the data properties. The graphical version of the model does not contain any inferred or inherited properties on the entities nor equalities with other ontologies. The representation of the model in OWL can be found in appendix A.

3.2 Entity selection

The Software Energy Consumption model provides a basis of the entities that are required to define the energy consumption data of software, and the environment of which the measurements are taken from. As such the entities provide a general basis and extra model additions can be used to further specify the used concepts.
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

Figure 3.1: Model Representation - The Software Energy Consumption Ontology presented in a UML class diagram.
3.2 Entity selection

The entities that are supported by the model are listed within this section. Each entity is attributed with its definition, rationale, attributes and properties, and its context inside the model (e.g. its parent and children).

3.2.1 Environment

The environment is a generalized concept used to describe a collection of resources in which an application can be executed. The entity is strictly used as an abstract definition which is further defined in software and hardware environment concepts.

Entity Context

```
<table>
<thead>
<tr>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Environment</td>
</tr>
<tr>
<td>Software Environment</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
```

Execution

The ‘execution’ entity represents data related to the finished execution of a software application. The entity is the collection of the workload, application, and environment and links this to the collected measurements and runtime attributes. Using the execution multiple different test runs of an experiment can be identified and distinguished.

Object Properties

- Execution Workload.
- Hardware Environment.
- Measurement.
- Software Application.
- Software Environment.

Data Properties

- Timestamp, the data and time when the execution was started.
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

- **Duration**, the amount of milliseconds taken to complete the full execution of the given workload.

Entity Context

```
   Execution
```

3.2.2 Execution Unit

The execution unit is a logical container within the source code in which functions can be defined. Depending on the underlying language the container can be classes, packages, or files. The concept provides an intermediate layer between the functions and modules in which they are defined.

Object Properties

- **Function**. An execution unit is defined as a container for different functions.
- **Module**. Inherited from Program Construct.

Entity Context

```
  Program Construct
  |
  Execution Unit
```

3.2.3 Execution Workload

The data provided by an application can depict an unique execution of a program. The specific set of instructions can lead to a distinct execution of as program, making it important data when an execution is analysed. The workload entity allows for reuse in multiple experiments and identifying the executions run based on a specific workload.

Object Properties

- **Execution**. The data given to a program will cause a specific execution of a profiled application.
3.2 Entity selection

- **Data.** The raw format of the data that is used as input for the program.

- **DataType.** A description of type of data that is used as workload. This can be for instance an “SQL Query.”

**Entity Context**

```
  Execution Workload
  ...
```

### 3.2.4 Function

In order to support different programming paradigms, the term function is used to defined the functions (or methods) within a code-base. The function entity can be used to defined the functions of a class, or within a source file depending on the paradigm used.

Although no data properties are currently added to the function entity, it can be extended with additional program metrics if there is need for these metrics.

**Object Properties**

- **Function.** Often functions are used to provide reusable code in different parts of the code base. As such functions often call different functions to perform more complicated tasks. This interrelation is used to defined these relations.

- **Execution Unit.** Link to the unit in which a function is contained, such a class or file.

- **Module.** Inherited from Program Construct.

**Entity Context**

```
  Program Construct
  | Function
  | ...
```
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

3.2.5 Hardware Environment

The hardware environment is a description of a collection of physical (or virtual) resources that make up the platform. Software applications can be executed within this environment, and can use the resources provided by this environment.

Note that the environment is specific to a single configuration. Therefore when the configuration changes the environment individual would also be different, making sure there is a distinction between the environment used in different experiments.

Object Properties

- **Execution.** Multiple program executions can be performed in a specific hardware environment.

- **Hardware Resource.** A hardware environment is made up of multiple components. These components further define the abilities and characteristics of an environment.

3.2.6 Hardware Resource

The resources within a specific hardware environment can influence the execution of a program. As these resources provide a role in the energy consumption it is important to classify these components individually.

The hardware resources can be used to model the different hardware components, physical or virtual, such as CPU or memory and connect the components to an hardware environment.

Object Properties

- **Hardware Environment.** Resource are components that collectively make up an environment. Resources can be defined as model specific individuals.
3.2 Entity selection


**Data Properties**

- *Property*, inherited from Resource.

**Entity Context**

```
Resource
  Hardware Resource
    ...
```

3.2.7 Measure

The measure defines the information that is observed from a resource within an execution. It contains the descriptive information on how the measurement is defined and how it can be used.

**Object Properties**

- *Measurement*. The observed values of this measure for a resource during a given execution.

**Data Properties**

- *Name*. The name of the measure that is defined.

- *Unit*. The format of the observed value, for instance Joules or Watts.

**Entity Context**

```
Measure
  ...
```

3.2.8 Measurement

One of the requirements for the model is to include measurements of an application execution. The measurement entity is defined as an observation of a property of a resource at a given point in time during an execution.
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

This means that the entity is used for dynamic, time dependant, data about a resource within a certain execution.

Object Properties

- **Resource.** The resource defines the object from which a measurement is taken and is required in the definition of a measurement.

- **Execution.** Measurements are taken during an execution, which can link the measurements to the program being executed or the data being processed.

- **Measure.** The measure defines the information of the measurement that is taken, such as the name and unit the value is defined in.

Data Properties

- **Timestamp.** The timestamp specifying the point in time that the measurement was taken.

- **Value.** The value that is observed.

Entity Context

```
Measurement
...
```

3.2.9 Module

A module is a part of an application that is used to perform a specific task. It has become common to reuse pieces of software in multiple applications, the model should be able to identify these components by defining them explicitly. In turn, reused modules can be used in future studies to investigate the comparative consumption and efficiency compared to different versions or similar modules.

Object Properties

- **Program Construct.** A module consists of certain constructions that define the code-base of the module itself. For instance the language or methods within the module
3.2 Entity selection

- **Software Application.** A software application is divided into a single or multiple modules that collectively form an application.

Data Properties

- **Version.** A value that identifies the version of the module.

Entity Context

```
    Program
    └── Program Library
```

3.2.10 Operating System

The hardware environment is managed by an operating system that, typically, links to the software environment. As the Operating System can have a significant impact on the performance of an execution and therefore qualifies to be defined as an entity itself.

Object Properties

- **Measurement.** Inherited from Resource.

- **Software Environment.** Inherited from Resource.

Data Properties

- **Property.** Inherited from Resource.

Entity Context

```
    Resource
    └── Software Resource
        └── Operating System
```

...
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

3.2.11 Program Construct

Some methods for the estimation of software energy consumption analyse properties of the program. The program construct is an abstraction for the construction elements that make up a module, such as the programming language(s) and functions.

Object Properties

- **Module.** The module in which the construct is used.

Entity Context

- Program Construct
  - Execution Unit
  - Function
  - Programming language
  ...

3.2.12 Program Library

A library is a pre-defined set of instructions that can be imported and used by an application to perform a certain function. The workings of a library cannot be directly changed by the actual program itself (with the exception of influenceable behaviour without adjusting the source).

Object Properties

- **Program Construct.** Inherited from Module.
- **Software Application.** Inherited from Module.

Data Properties

- **Version.** Inherited from Module.

Entity Context

- Module
  - Program Library
  ...

30
3.2.13 Programming Language

There is a wide selection of programming languages that can be used to write software, not only with syntax but also with the underlying library. The differences between these languages are often not easily distinguishable. As such the modelling of the language can become a valuable asset. Not only can default or additional language-specific information be added, but with enough data the effects between languages can be studied.

Object Properties

- *Program*. Inherited from Program Construct.

Data Properties

- *Version*. An identifying value for the version of the language the individual represents.

Entity Context

```
Program Construct
  Programming Language
    ...
```

3.2.14 Resource

Similar to the *environment*, the resource entity is a generalized concept to describe a resource within an environment. The underlying entities further define the resource by the separation into hardware and software.

Object Properties

- *Measurement*. The measurements done on this resource within a specific execution.

Data Properties

- *Property*. A value that provides static information about the resource.

Entity Context
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

3.2.15 Software Application

An software application is a generic definition for any type of program that can be executed. This includes programs that provide an interface to end-users or provide functions such as databases or utilities.

Object Properties

- **Module.** A software application is, within the model, defined to exist from one or more modules.

- **Measurement.** Inherited from Resource.

- **Software Environment.** Inherited from Resource.

Data Properties

- **Property.** Inherited from Resource.

Entity Context

3.2.16 Software Environment

The software environment defines the resources that are used or available for an execution within the software layer of the infrastructure. Analysis results for an execution can be strongly influenced by other resources running on the experimental machine.

Object Properties
• **Software Resource**. The software environment is composed of multiple software resources.

• **Execution**.

Entity Context

```
Environment
  Software Environment
  ...
  ...
```

### 3.2.17 Software Resource

A software environment contains different resources that can or are used that influence the performance. These resources provide valuable information on the collected data and its interpretation. A software resource is defined as any resource which only contains of software and is not contained within hardware components. However, it is possible for a resource to be specific for a distinct hardware resource.

**Object Properties**

- **Software Environment**. The environments in which the software resource is contained.

- **Measurement**. Inherited from Resource.

**Data Properties**

- **Property**. Inherited from Resource.

Entity Context

```
Resource
  Software Resource
    Operating System
    Software Application
    ...
  ...
```
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

3.3 Model connection

The software consumption ontology defines all the required entities itself. As such it does not require external ontologies to further specify any of its entities. The reuse of existing ontologies can provide valuable advantages, such as the reuse of data contained in other models or provide extensive details in the areas covered by the external ontology.

The ontologies found in section 2.3.2 provide more specialization in the interest areas that overlap with the software energy consumption model. The inclusion of these models can improve the validity and impact of the Software Energy Consumption Model (SECM). Therefore, it is beneficial to see the relevance of the found models, and how they are connected to SECM.

In total the search was conducted for three specializations contained within the model and identified six different models. As the connection to the other models is not the focus of this study, only the potential overlap, concerns, and advantages for linking are discussed.

3.3.1 Open Multi-Net

The Open Multi-Net ontologies defines various concepts which can be sued to define the hardware environment of an experiment. as the Open Multi-Net is focused on the entire infrastructure, the available data can be analysed for extensive analysis of the environment impact.

Although the SECM can model the relevant components in its environment, it is limited to the direct components. The inclusion of Open Multi-Net can be highly valuable, allowing the modelling of an extensive infrastructure.

- The concept Component has a clear definition and (sub)partitioning on different hardware components. The SECM:hardware resource has the same goal, making it a perfect candidate for reuse. This will allows to include OMN infrastructure information into experimental set-ups for an extended description of the environment.

- The Environment concept in Open Multi-Net is used to define the space in which the hardware components work together. As the OMN model has the focus on
3.3 Model connection

hardware infrastructure, the concept is closely related to the $SECM:hardware$
$environment$.

3.3.2 Linked Source and Documentation Ontologies

Although the ontologies for the source and documentation as proposed in [12] are generated by an application, it contains a default model backbone with the generalized concepts used within the model.

The model backbone used by the generated ontologies is considered as the same model. Using this connection, the entities that exist in both the ontologies are treated as the same entity.

- The $Class$ concept can be mapped to the $SECM:Execution Unit$ as they share the same idea. The $SECM:Execution Unit$ is a more abstract version of the entity for usability in functional languages.

- The $Method$ is identical to the $SECM:Function$ having both the same type of data they attribute.

3.3.3 KOontR

The KOontR model is part of an approach to improve software reuse. The idea is to define a knowledge-base of different application components that can be reused. Using this approach a database is constructed where interested parties can extract information about used licenses, experienced developers, or usages of components [14].

The data contained within the SECM can add an extra dimension to possible selection of reusable components in the future. Future study is required to define the usability and effectiveness of this information as well as reconcile the models themselves.

3.3.4 EvoOnt

The EvoOnt proposed in [15] separates the ontology into three parts. Only the Software Ontology Model (SOM) sub-ontology shares a similar interest with the Software Energy Consumption Model. However, a detailed definition on the ontology is missing and only the OWL source document is referenced.
3. THE SOFTWARE ENERGY CONSUMPTION MODEL

The possibilities for linking both ontologies is not researched, as the model definitions are missing, a crucial part for model reconciliation is unavailable. The definitions within the references OWL model provide disjoint ambiguous concepts where the difference is left unexplained. Thus this study will not further define any connection between the models.

3.3.5 TwoUse

The TwoUse approach defined a hybrid model that combines UML and OWL, and allows transformation to both type of models. Applications that are defined using this method have the ability to transform into a complete OWL model of the application. As such the inclusion can be valuable.

However, the method that is proposed defines the entities in OWL as classes. As such there is a difference in granularity with the Software Energy Consumption Model, which defines a class as an individual. More in-depth analysis is required to further specify the inclusion fitness of this approach.

3.3.6 Software Measurement Ontology

The goal of the defined ontology is the inclusion of measurements taken of a software application relating to the energy consumption during an execution. The Software Measurement Ontology (SMO) can provide a valuable connection to the model for the measurement data.

The SMO is an extensive studied model incorporating various standards for software measurements, and as such can be used to further specify the measurements used in experiments. Besides the gained value of the data, it can additionally prevent misconceptions and inspire new relevant types of measures for software energy consumption.

The entities measure and measurement within the Software Consumption Ontology are inspired and share the purpose with similar names entities in SMO.

- The SCO:Measure is a simplified concept compared to the SMO version, as the full definition of the measure itself is not required in the typical usage of the model. Only the essential definition of a measure are taken, and using SMO it can be extended to define the measures in depth.
3.3 Model connection

- The *SCO:Measurement* couples to both the *SMO:Measurement* and *SMO:Measurement Value*. Although a distinction is made between the ‘action’ and ‘result’ of the measurement process, this separation is irrelevant for the consumption data.

It is important to note that the *SMO:Measurement* has been defined as: “a set of operations having the object of determining a value of a measurement result []” [18], whereas the *SCO:Measurement* incorporates the result within itself.
3. THE SOFTWARE ENERGY CONSUMPTION MODEL
4

Model Case Study

4.1 Evaluation Case

The Software Energy Consumption Model is evaluated using the data of an existing experiment. The data from this experiment are explored and mapped to the entities in the model. The experiment, a case study, is taken from the PhD dissertation “Energy-Efficient Software” [6]. Both the case study description as the datasets of the measurements, which are publicly available[^1] are used for the evaluation.

The evaluation will focus on the collected data of the MySQL case study described [6]. Any results of the analysis on the data will not be included in the evaluation.

Two main points are used to determine the ‘fitness’ of the data model. First, to ensure a location for all related data, the model must be able to include the entire dataset in the model without any truncation. Secondly, the original data should be identifiable within the model and allow the recreation of the original datasets. The translation back to the previous format can be used as a simple confirmation that all semantic information is translated, that is, no inferred data is lost within the Software Energy Consumption Model.

4.2 Data Identification and fitting

This section will place the data from the case study within the defined Software Energy Consumption Model (SECM). The model itself can be found in appendix A from which[^1]http://www.s2group.cs.vu.nl/green-lab/
4. MODEL CASE STUDY

individual parts will be described in this section.

The model is build in layers, starting at the hardware layer (hardware environment). The first descriptions of the environment is the test machine itself [6, p.47], which consists of a Dell PowerEdge SC1425 server. The machine will be defined as a Hardware Environment under the same name.

Next the resources of the machine are added to the model. Besides the basic components of the machine, such as the CPU, we also identify the instrumentation equipment as hardware resources. The instrumentation consists of three different components which are attached to the various components.

The data is restricted to only the information listed in the experiment description. Each of the components are modelled as separate unique individuals, allowing the tracking of possible equipment specific properties within an experimental setup.

```xml
<owl:NamedIndividual rdf:about="&secm;SEF_Dell_PowerEdge_SC1425">
  <rdfs:label>Dell PowerEdge SC1425</rdfs:label>
  <rdf:type rdf:resource="&secm;Hardware_Environment"/>

  <secm:has_hardware_resources rdf:resource="&secm;Dell_power_Supply"/>
  <secm:has_hardware_resources rdf:resource="&secm;Infineon_1GB_DDR2−333_1"/>
  <secm:has_hardware_resources rdf:resource="&secm;Infineon_1GB_DDR2−333_2"/>
  <secm:has_hardware_resources rdf:resource="&secm;Infineon_1GB_DDR2−333_3"/>
  <secm:has_hardware_resources rdf:resource="&secm;Infineon_1GB_DDR2−333_4"/>
  <secm:has_hardware_resources rdf:resource="&secm;Intel_E7520_Chipset"/>
  <secm:has_hardware_resources rdf:resource="&secm;Intel_Xeon_CPU_1"/>
  <secm:has_hardware_resources rdf:resource="&secm;Intel_Xeon_CPU_2"/>
  <secm:has_hardware_resources rdf:resource="&secm;Maxtor_7L250S0_1"/>
  <secm:has_hardware_resources rdf:resource="&secm;DAQ_1"/>
  <secm:has_hardware_resources rdf:resource="&secm;DAQ_2"/>
  <secm:has_hardware_resources rdf:resource="&secm;Wattsuppro"/>
</owl:NamedIndividual>

Figure 4.1: Fitted hardware environment - The hardware environment used in the experiment fitted for SECM.

The resulting individual for experimental the hardware environment can be found in figure 4.1. Line 2 and 3 in the figure list the name and type respectively of the individual within the model. The components such as CPU and memory modules are listed in the
4.2 Data Identification and fitting

lines 5-13. Finally, the instrumentation equipment consisting of two Texas Instruments Data Acquisition Boards (lines 15 and 16) and a single Wattsup PRO meter (line 17) are listed in the individual.

The resources are modelled the same way, as such only one of the resources will be described in more detail. Figure 4.2 lists one of the memory components that is used within the hardware environment. Line 1 shows that the module as been given an unique ID ("Infineon_1GB_DDR2-333_1"), this allows the four different modules to be defined separately. Next the information from the name, such as the size and clock speed, is listed as properties of the resource (lines 5-7).

```
1 <owl:NamedIndividual rdf:about="&secm;Infineon_1GB_DDR2-333_1">
2   <rdfs:label>Infineon 1GB DDR2-333</rdfs:label>
3   <rdf:type rdf:resource="&secm;Hardware_Resource"/>
4
5   <secm:has_property rdf:datatype="&xsd;string">Interface: DDR2 SDRAM</secm:has_property>
6   <secm:has_property rdf:datatype="&xsd;string">Memory Clock: 333 MHz</secm:has_property>
7   <secm:has_property rdf:datatype="&xsd;string">Size: 1GB</secm:has_property>
8 </owl:NamedIndividual>
```

**Figure 4.2: Fitted memory module** - One of the memory components used in the hardware environment.

The software environment is modelled the same way as the hardware environment. First the collection individual is created with an unique name. Followed by linking all of the software resources present. The identified resources are the operating system (line 5), monitoring applications (lines 6-8), and the program that is tested (lines 9-10).

The experiment was done using both the default version (Vanilla) and a modified version containing code instructions for the monitoring tools. As there are differences in the code-base of both version, the applications are modelled as separate applications.

The experiment description contains no further information on attributes of the resources, such as version numbers, excluding these variables from the model. Additionally, the application is not further defined for its internal functioning, as this data is not provided by the case description.

The environment is now fully defined within the model, as well as the application(s) that are tested in the experiment. Next the executions can be added to the model.
Three different scenarios are executed during the experiment where each is repeated 10 times. For clarity only the first two executions of each scenario is fitted in the model.

The three scenarios are defined in the datasets as vanilla-, original-, and modified-mysql. These names are taken over for the identifiers for each of the executions, setting ‘VM_Run1’ for the first run of the ‘Vanilla Mysql’ scenario which is shown in figure 4.4.

The data properties of the execution, such as the time stamp and duration (in milliseconds), are extracted from the datasets and added to the model (lines 4-5). Next the previously specified environment is added (lines 9-10) as well as the application this execution targets (line 7). Finally the workload, in this case a specific SQL Query, is added to the model and linked to the execution (line 8).

As a final step the measurements are added to the model. The data contains 37 different types of measures, each consisting of many records that are recorded over time, however some measures are similar, although focusing to different resources.

The Measure entity in the model can be used to describe the different measures that exist in the dataset. Five measures are taken from the dataset that can be grouped together as an example for the model fitting:

- **12 Volts wattage.** The consumption in watts per second for the 12 volt connection to a resource.

- **5 Volts wattage.** The consumption in watts per second for the 12 volt connection to a resource.
4.2 Data Identification and fitting

Figure 4.4: Fitted software execution - An execution done in the experiment for the Vanilla Mysql scenario.

- **% time CPU sys.** Percentage of the time the total CPU cycles spend in the kernel within the process.

- **% time CPU usr.** Percentage of the time the total CPU cycles spend outside the kernel in user-space within the process.

- **Memory used.** The total amount of bytes of memory consumed by the application.

The measure for the 12 volt connectors is shown in figure 4.5. The other measures follow a similar construction with different values for the data properties. The *name* attribute of the measure can be used to provide standardized naming for the measurements, in the context of this evaluation only a small definition is used.

Figure 4.5: Fitted measure - The watts consumption for 12 volt connection measure fitted to the model.
4. MODEL CASE STUDY

Now that the measures are defined, the only thing left are adding the actual measurements to the data model. Each measurement can be added with a timestamp and a value, and is then added to an execution (the run it was taken from), the resource, and measure.

As the measurement is connected to both the resource and the measure, recurring measures such as 5 Volts wattage can be reused and identified using the resource that is connected. For instance, in the dataset both the watts for the Motherboard as the HDD are recorded. Both are the same measurement with only a different resource as a target.

4.3 Discussion

The model fitting used the data from existing experiments and shows where the data belongs and how it can be fitted. The resulting model was able to contain all of the information provided by the experiment details. Most of the data was able to fit logically in the model, however there is still room for improvement.

The model provides a generic approach for the modelling of the resources used in the experiments. Although this method allows for any type of resource to be added to the model, the identification of specific components, such as the CPU, can become tricky. An extension to the model could provide a specific definition for some of the entities. A link to ontologies that define these attributes, such as the Open Multi-net, could improve the value and usability of the model.

Furthermore, the resources in the environment have additional static information that (typically) does not change over time, such as size or clock speed. Although the model provides an data property to define these values, it is limited to the storage of a single value, even though most of these attributes are more fitting as a key/value pair. Possibly this can be added to the inclusion of further defined resources, and provide data attributes for specific components.

Another limitation found during the case study is the instrumentation of the power modules. The experiment contains various instrumentations that provide information on the performance and energy consumption on specific components. Although the instrumentations can be placed within the model, the resource the instrumentations observe can only be defined in a description.
However, the case study executed shows that the criteria defined are satisfied. First of all because all the data can be presented in the data model. During the study the data was collected from both the datasets and the description. We argue that the model even provides more information than the previous set, due to the fact the environment is contained within the model, instead of described in an article which is not attached to the dataset.

The second criteria is also satisfied. The insertion of the data in the model was done with careful precision so that all data values are unique in their given context, a good example of this are the measurements which are unique by the type of measure, the target resource, execution, and time it was taken. Therefore, we state that, using the fitted model, the original datasets provides can recreated.

As both of the criteria are satisfied we state that the model is able to fulfil the first purpose it was given: “the storing and sharing of experiments, with the measurements taken, the environment the experiment was executed in, and the software application that was analysed.” Furthermore, the semantic relationships defined within the model form the basis to discover possible more complex relationships within the dataset. Using OWL, various semantic reasoners are available which can be utilized to infer undefined (and possibly unknown) relationships within the model.

Even though we state the model is able to fulfil the given objective, areas still exist for further development. The model is defined as a foundation for the storage of software energy data, as such future studies are required for further build upon the laid foundation.

4.4 Threats to validity

The case study performed on the Software Energy Consumption Model (SECM) shows that the objective specified for the model is fulfilled. Although the initial definition and current case study are to prove the validity of the conceived model, there are a few points which can pose a threat to the validity.

- **No complete model evaluation** - The case study used for the evaluation was from a predefined experiment, of which the data does not match the granularity SECM
allows. The underlying definition of the tested software, the modules and program constructs, are not part of the experiment and thus are missing from the evaluation.

- **No Usability Experiment** - The model is evaluated using a case study, where the data is fitted into the model. No experiments are done with the resulting model that prove the usability of the model in experimental contexts.

- **Model connections** - This study contains a short study into existing models which can be (partially) reused in SECM. There is a high likelihood that models are missed which could provide valuable information and improve the impact of the model.

- **Domain Experts** - According to the steps for defining ontologies described by Suares-Figureoa et al. [21], the usage of domain experts is an important step. Even though advice from experts was used in the conception of the model, no extensive discussion were performed with domain experts.
Part II

Data Visualization
Visualization Background

5.1 Introduction

Software applications are known to have the ability to generate data at prodigious rates. The analysis of software performance follows this pattern, and depending on the parameters can create vast amounts of data for applications of any size.

Analysing the raw data from applications can be cumbersome. Graphical representation adds a new dimension to the data, which can improve the analysis of the data. Using graphical tools properties of the dataset, such as relations, clusters, or hotspots can be quickly found and analysed further.

This chapter will analyse current techniques used for visualizing application data that can be used for measuring its energy consumption.

5.2 Context

The visualization of data is a mature topic. Many techniques have already been developed and is still an active research topic. Various conferences are dedicated to data visualization, and present a platform for new methods and research. An example is the UpSet technique by Lex et al. [22] submitted to the IEEE VIS conference in 2014, which uses an approach based on sets and their interactions.

Data visualization techniques are in demand by companies to gain insight into their data. Dashboards are often given with products to provide such insight, some companies

\(^1\)http://ieeevis.org/
adapted their business model to providing and consult on these tools.

In relation with software energy efficiency, the amount of tools that are used are limited. Papers discussing the energy consumption often limit themselves in strictly using scatter-, bar, and line-plots.

5.3 Objective

Software energy efficiency is an actively discussed subject are simple methods are applied in visualizing it. The main aim of this research is to find novel methods to analyse data relating to the energy consumption. The research question, as established in the introduction (section 1.3), is defined as:

“How to visualize the most relevant entities and relationships of the data model for software energy efficiency?”

Our approach to this research question is the analysis of two different methods for analysing the energy related data. First use a visual analysis tool for exploring software energy efficiency data in chapter 6 followed by a graphical representation combining both software and energy consumption in chapter 7.

Each method is paired with an individual experiment showing the potential usage and relevance for each method. The results of the experiments are discussed within their respective chapters.

5.4 Existing studies

The visualisation of the consumption of energy is, on its own, a complex matter. Research such as done by Gustafsson [23] investigate for visual presentation of power, instead of providing static information in watts.

As the relationship between software and energy is an open research question, the lack of available research in the subject is not surprising. However, novel techniques are being developed for visualizing energy consumption in the context os software.

An interesting approach is presented by Carção [24], who suggests a system for detecting “red code smells” during the software development process. The proposal uses a database of good (“green”) and bad (“red”) coding practices in terms of energy
5.4 Existing studies

consumption, which are defined in an experimental setting. Carçao envisions that the resulting program “[..] can identify red smells in the program and try to recommend green refactorings to improve the energy consumption.” \[24\]

A IEEE Working Conference on Software Visualization (VISSOFT) provides a platform for visualization techniques that help understanding software applications. Source-Vis is presented by Anslow et al. \[25\] at the 2013 conference, the technique focuses on the visualization of software during the maintenance process. It provides various information about the classes, such as the size and complexity. Adding energy related information, the tool could additionally visualize the energy consumption and show energy hotspots.
5. VISUALIZATION BACKGROUND
Analysing Software Energy Efficiency Data

6.1 Introduction

The collection of data relating to software energy efficiency can result in large datasets. Manually analysing this data consume a lot of time, visual analysis of the data can help speed up the process and uncover new connections within the data. Many tools exist for the creation of graphical representations of data, however these are often a static view of a small subset.

In order to see how different tools can attribute to the analysis of data relating to software energy efficiency a visual tool is analysed for its usability. The tool selected for this analysis is SynerScope, an tool which can be used for visually analyse datasets.

The tool is developed by the equally named company SynerScope, and provides different white-papers demonstrating the capabilities of the tool. It differentiates itself from similar tools by allowing dynamic analysis of the data, providing different methods for visualizing the data, and the ability to link selected data between different views.

This chapter will first discuss the data that was used in the experiment, and the pre-preparation of the data. Next the performing of the experiment itself, showing interesting points found in the data before discussing the results.
6. ANALYSING SOFTWARE ENERGY EFFICIENCY DATA

6.2 Preparation

The experiment for the SynerScope tool uses the same data taken from “Energy Efficient Software” [6]. The experiment was performed on two applications: an Apache HTTP server, and a MySQL database engine. Each application had an original version, as well as two adjusted versions where one factor (or variable) was modified. For each version the experiment was repeated 10 times in the same setting, to validate the results.

Using R, an environment for statistical computing, the data is combined into a single dataset. Three additional fields were added to the dataset based on the original data: TimeFromStart the amount of milliseconds since the start of the experiment, CpuSysUsr the total time of the CPU spend on system or user functions, and TNorm an percentage indicating how much of the experiment is completed at a given point in time.

The dataset, consisting of 41 data points with each approximately 30,000 observations, is extracted and imported into the SynerScope tool. The tool requires the data to be separated into two categories: nodes and links. The nodes can be seen as categories, or grouping of information, whereas the links connects them together. For example, in the case when analysing traffic information, the source and destination locations can be nodes, whereas the roads in between the locations are the links containing the traffic information of a given moment in time.

The used dataset consists mainly of dynamic data, that is, the observations of software and hardware resources in a given point in time. Two different nodes are taken from the dataset, which are connected with a single link. The first node groups based on the case study (mysql or apache) and the unique ID of the experiment instance, the ‘runid’. The second node, as well as, the link contain all of the data points, allowing the grouping and bundling of data points during the visual analysis.

6.3 Experiment

The experiment first focusing on both experiments individually, starting with MySQL and followed by the Apache Server. Each application has been executed in three scenario’s: vanilla (taken without any instructions), original (with instructions before optimization), and modified (with instructions and optimizations) [6], pg. 49]. The experi-
6.3 Experiment

The MySQL experiment will address the scenarios as MV, MO, and MM for the MySQL application and AV, AO, and AM for Apache application respectively.

MySQL The MySQL experiment applies the practice of using “efficient queries,” which will analyse the effect of including the ORDER BY statement. The scenarios MV and MO have the ORDER BY statement in the used experiment, which was removed in the MM scenario.

The analysis of the MySQL experiments is started with the data relating to the energy consumption. The energy was taken from power meters with an interval of 1 second. Using the bundle view in SynerScope, the watts were separated into 10 equally sized bins (groups) by the program. Each group covers a range of 31.1 watts and the usable groups were defined as $0 < 31.1$, $154.4 < 185.5$, $185.5 < 216.6$, $216.6 < 247.7$, and $\geq 247$, no values were reported between 31.1 and 154.4 leaving these 5 groups out of the experiment.

As SynerScope is a tool for dynamic analysis of the data, a subset can be selected within the views. Figure 6.1 shows the grouping of the wattage on the left, with the scenario and individual experiments for MySQL on the right.

![Figure 6.1: Energy Usage MySQL - Grouping of the wattage usage of the MySQL experiments, with observations (from left to right) of $< 247.7\, \text{w}$, $247.7–216.6$, $216.6–185.5$, and smaller groups with values $> 185.5$ with are not highlighted.](image)

The selection of the group with the highest wattage (on the left) shows that MM is accountable for the highest energy readings. Only experiments from the MM scenario show up in the highest consumption group. The second (middle) and third (right) consumption group show a more even distribution amongst all the scenario’s.
6. ANALYSING SOFTWARE ENERGY EFFICIENCY DATA

When looked further into the energy consumption of the MM scenario, two spikes can be identified. These spikes are visible on figure 6.2 which shows the consumption in watts over time for the MM scenario.

Figure 6.2: Energy Spikes Identification - The grouping of the energy consumption (left), the usage of energy over time (middle) and the processor usage over time (right) for the Mysql Modified scenario. The selected values (in orange) correspond in all three views.

The figure shows a pattern of decreasing energy consumption from the start to the end of the query. However, between the execution of the first and second, as well as, the second and third query a spike can be seen in the observed watts consumed.

During these spikes the memory consumption of the application changes. Figure 6.3 shows the free and cached memory of the application. Around the spikes in energy consumption, the amount of memory that is cached or used by the application shows a noticeable drop.

Figure 6.3: Memory Usage Pattern - The consumption memory over time by the Mysql Modified scenario. The view on the left shows the cached memory, the middle shows the amount of free memory, and the right shows the used memory by the application.

The high consumption during the cleaning protocol of the memory might indicate
some garbage collection (memory no longer used) is performed after a query completes. Interestingly enough, this behaviour is not seen in the other scenarios.

Looking at the memory patterns for the MV and MO scenarios, as shown in figure 6.4, it shows a different usage in types of memory. Both scenario use roughly the same amount of memory during the experiment, with the difference of MV using more cache memory and the MO scenario using its own allocated memory.

![Figure 6.4: Memory Usage Comparison](image)

The switch between the cached memory and the used memory is interesting, as according to the description no changes were made in the functioning of the application. The difference in memory usage however seems to have no effect on the actual energy consumption, as shown in (C) of figure 6.4. This might state that there is no significant difference in the usage of normal or cached memory.

**Apache HTTP Server**  The experiment for the Apache HTTP Server focuses on the practise of putting applications to sleep when not used. The practice is applied by removing all invocations to the `sleep` method in the code of AM, preventing the application being placed in a sleep state.

The experiments with the Apache HTTP Server show a more uniform dataset, with the vanilla scenario (without code instructions) as the outlier. Interesting enough, the scenario shows a distinct pattern of growing memory for each instance of the experiment run. The pattern is shown on figure 6.5, the points in orange are taken from the AV scenario.
6. ANALYSING SOFTWARE ENERGY EFFICIENCY DATA

**Figure 6.5: Apache Watts and Memory** - The view shows the grouping of the watt consumption (left) and the memory consumption (right) of the Apache HTTP Server.

The same pattern is repeated 10 times in the figure, with the first experiment listed on the bottom. The pattern could indicate that some of the memory used by the Apache application is not cleaned after its done processing, often referred to as a memory leak. Especially near the end of the execution the memory usage spikes, indicating that some part of the application has some unexpected behaviour.

### 6.4 Discussion

The impact of software on the energy consumption of ICT systems is still an active research topic. Although research has proven the impact of software on the hardware consumption, the actual relationships are still obscure. As not all the factors are known, the exact information required to analyse the energy impact of software is undetermined.

As a result datasets contain numerous types of observations that need to be analysed together. For the experiment we used the publicly available results taken from experiments performed by Procaccianti [6]. The dataset taken from this has approximately 1.23 million values spread across 41 different types of observations.

The dataset was visually analysed in the SynerScope tool without taking the results of Procaccianti in regard, focusing on an unbiased exploration of the data using the tool. Although the program only provides limited tool for preprocessing the data, once its loaded it can be used and updated with ease.

The dynamic analysis of the tool allows for analysing individual data points in different views. In the case of figure 6.2 the highest consumption of energy can be
compared to both the usage of energy over time and the activity of the processor. Due to the immediate selection across all views, it shows that the high consumption of energy is not directly related to the processing power of the experimented application.

Furthermore, the side by side comparison of different scenarios, as shown in figure 6.4, allows for analysing the difference between version of the same application. For example, the figure shows two scenarios (Original and Vanilla) of the experiment conducted on MySQL. Although the patterns show similarities, there are differences in the amount of memory used by both scenarios, which turn out to have no effect on the energy consumption.

The ability for dynamic visual analysis provided by the SynerScope application can be very interesting researchers, but also to application developers. Research can benefit from the ability to analyse large quantities of data and link data points between different view, which can lead to the discovery of previously unknown relationships in the data.

Application developers whom are interested in the development of energy efficient software can use the tool to analyse the impact of application change for their energy consumption. Although the usage requires some automated preprocessing of the data, the ability to correlate data from different versions of the application can visualize problematic locations within the application in terms of energy usage.
6. ANALYSING SOFTWARE ENERGY EFFICIENCY DATA
Graphical Representation of Software and Energy

7.1 Introduction

Flame graphs are a special type of visualization conceived by Brendan Gregg [26], a computer performance specialist, to visualize code paths that are active on the CPU. Taking data from a profiler, the stack traces are presented as blocks stack upon each other.

Figure 7.1 shows an example of a flame graph displaying the code paths, the order in which functions within the code are executed, taken in MySQL [26]. The width of an block shows the occurrence in the profiling logs. The highest block in the graph represents the function that is being executed by the processor.

The colours used in the example flame graph have no meaning and are purely cosmetic. In some cases the colours have been used to show differences between the programming language that is used.

Since its first appearance in 2011, adaptations have been made to show other types of information, such as Memory Flame Graphs to show the amount of memory that is allocated by different methods, or Off-CPU Frame Graphs to show which methods are often blocked (waiting for something to continue processing).

Bezemer et al. [27] use an adapted form, differential flame graph (DFG), to visualize differences between different performance profiles. The DFG shows the flame graphs of the two profiles, and uses a third to visualize the differences between the graphs.
7. GRAPHICAL REPRESENTATION OF SOFTWARE AND ENERGY

![Flame Graph](image)

**Figure 7.1: Flame Graph** - An example of a flame graph with data from a profiling of MySQL [26].

The third graph allows for quick analysing of the difference (and effects) of the sampled profiles.

### 7.2 “Energy” Flame Graphs

The purpose of flame graphs is to analyse the performance of software. The energy information could provide useful analysis for efficient or inefficient methods in terms of their energy consumption.

The traditional flame graphs use the amount of occurrences of a specific stacktrace, the function in the source code being process with the path taken, to estimate which calls spend most time on the CPU. The width is therefore not directly related to the execution time of the profiled applications.

Simply linking the occurrences with the energy consumption is not feasible, as the occurrences are not linked with the moment in time they are taken. The traces used in the ‘normal’ flame graphs are sorted alphabetically and simply record the amount of times a specific trace is observed. Energy flame graphs require more information in order to link the tracing with the energy consumption. First of all, the stacktraces need to be recorded with the moment in time they are taken. Furthermore, the traces cannot be merged in the sense that, the time is the value that orders the stacktrace.

Additionally, the energy consumption of the system needs to be measured in intervals during the execution of the profiled application. The observed energy consumption can
then be tied together with the stacktraces which were executed during the profiling period.

7.3 Proof of Concept

Normal flame graphs can be generated by a publicly available Perl script written by the author, Brendan Gregg. The output of this script is a special image (SVG), that allows the user to zoom in on the stacks. However, as the underlying data, as well as, the required visualization is different, a new implementation was made for the proof of concept.

The proof of concept implementation is made using the Java programming language, using Swing [1] to visualize some of the components used in the visualization.

First an overview is given of the data input, and the processing of this data. Next, the primary visualization of the 'energy flame graphs' are discussed, followed by an extra mixture between energy- and differential- flame graphs which was added to the program.

7.3.1 Data Input

The proof of concept program has been designed as a standalone program, meaning that it (on purpose) does not take into account the model from Part I of this thesis. Instead the input of the program consists of two Comma Separated Value (CSV) files.

The input files can be used to store the energy consumption and stacktraces taken from the profiled application. The data is read from the input files and processed by the application. The program accepts the following values for each of the data types:

- **Energy Data.**
  - *Time*, the time the observation was taken in the form of the amount of milliseconds since January 1st, 1970;
  - *Watts*, value indicating the amount of watts consumed during the interval.

- **Stacktrace Data.**

[1] https://docs.oracle.com/javase/tutorial/uiswing/
7. GRAPHICAL REPRESENTATION OF SOFTWARE AND ENERGY

- **Time**, the time the observation was taken in the form of the amount of milliseconds since January 1st, 1970;
- **Thread Name**, the name of the thread the stacktrace was taken from;
- **Stacktrace**, the code path that is currently being executed by the thread.

**Energy Data** The programs considers energy is taken in the amount of watts in a specific interval. The interval is the time between the current and the previous value.

During processing the program will discard the first wattage that is observed, linking the time to the starting moment of the interval for the second value. In order to prevent the overlap, the starting time of an interval is incremented with 1 millisecond, making the times for the energy values inclusive.

**Stacktraces Data** Threads are a common phenomenon in current day programs that allow for different code paths to be executed simultaneously, that is, a function can start processing before another function is finished. These threads typically follow a different path through the source code, based on the given instructions.

In order for the program to differentiate between threads, the stacktrace is stored alongside the name of thread which is processing that particular trace. The program can use this information to group the data together, and visualize the path (or flow) a thread has taken through the source code.

The time the stacktraces are observed are also stored within the data file. Making it possible for the program to cross-reference these with the energy values.

The data is processed in two steps, preparation, and merging. The steps are further discussed below.

1. **Preparation.** The input is first split into three values: *time, thread, and stacktrace*. The stacktrace is split into *elements* each representing a function.

   The data is then grouped based on the value given for *thread*. The grouping makes sure that the stacktraces for each thread are not mixed together.

2. **Merging.** The elements are compared to the previous elements from bottom to top, eg. the first function called is compared first and the current processing function last. If the element is equal, the time-frame of the previous element is extended, when non-equal, the program creates a new entry.
7.3 Proof of Concept

An example is shown in figure 7.2 showing how the values are being set in the processed timestacks. The example shows four stacktraces \([A \rightarrow B], [A \rightarrow B \rightarrow C], [A \rightarrow D \rightarrow C], \) and \([A]\) for a single thread with timestamps 10, 20, 30, and 40 respectively. As function \(B\) is on the same depth with the same element for function \(A\) below, both entries are merged together. However, in the case of function \(C\), it has a different element below in both cases (at time 20 \(B\), and at time 30 \(D\)), and thus is not merged together.

The red lines in the figure illustrate how the timing is done in the merged stacks, the stack \([\rightarrow B \rightarrow C]\) at time 20 is different then the stack at time 30. As such the stack is executed within this interval. The time is then set as interval of time 20 to (and including) time 29.

Figure 7.2: Stacktrace merging - Illustration of the stacktracing merging done by the program. The left shows the input stacktraces, whereas the right shows the stacktraces after merging.

The normal visualization of the flame graphs takes a fixed width, and bases it on the amount of occurrences to specify the width (in percentages) based on the rest. The small stacks (with low occurrence) can be explored by enlarging the image (which is added by the script).

However, in the case of energy flame graphs this is unwanted as its based on the execution time. The profiling on an application can take a longer period of time and posing the fixed width of an image would require several enlargements to make it readable.

Instead the program visualizes the stacktraces, as opposed to making it an image. The width of each element is based on the time it spends being processed by the processor. In order to visualize the small elements (that finish quickly), the width are adjusted to a scale with a factor of 0.01 to 10.
Figure 7.3: Flame Graph Visualization - Example of the flame graph view of a profiled application.

The flame graph visualization in the proof of concept program is shown in figure 7.3. The following paragraph lists the different components that are shown on screen with their purpose, the numbers correspond to those shown in the image.

1. The visualization of the elements of the stacktraces. The small stacks are shown as lines as they processed in a short amount of time.

2. The cursor, allowing the selection of a specific element in the view. The red line can used to reference the time at the given location.

3. Information about the location of the cursor. On the top the currently selected function is shown, below is the time frame in which this function is processed, the third line shows the time that is shown at the location of the red line, and the lowest the wattage at that moment in time.

4. General information on the view, on the left it shows a line that represents 1 second of time, below that is the timeframe of the entire view. On the right is the scale that is currently used, as well as the frames per second (FPS) which shows how fast the application is drawing the screen.
5. The wattage that is measured at this moment in time. The scale on the left of the view shows how much wattage is used by the program at this moment.

6. A list of all the threads that are available for viewing. The last overview is the differential energy flame graph discussed in subsection 7.3.3.

7. The time-line based on the stacktraces. The filled part of the time-line shows the currently visible stacks based on the total amount available for this thread. In figure 7.3 all stacktraces are shown, so the bar is filled completely.

### 7.3.3 Differential Energy Flame Graphs

The differential energy flame graph (DEFG) is based on the idea from Bezemer et al. [27]. In their article the authors use a differential flame graph to visualize the difference between different implementations of the same code, which allows a visualization of the changes in the processing after adjustments.

The DEFG takes over the idea of showing two flame graphs side by side and comparing the differences. However, while the Bezemer’ differential flame graphs compares different implementations, the DEFG compares the processing time (CPU Time) and the amount of energy consumed. Allowing the developer to estimate that the time spend processing is roughly in comparison ot the energy it spend.

An example of a DEFG is shown in figure 7.4. The figure shows on the top the time spend processing a specific path in the code (stacktrace), the same trace is highlighted in red on the bottom where it shows the estimated consumption in joules.

The example shows that the selected stacktrace “WorkerThread:run→WorkerThread:remove→implementation.FineGrainedList:remove” represents 12.85 percent of the total processing time, while it only attributes for 9.26 percent of the estimated energy consumption. Using this method it could show the efficiency of certain stacks in terms of processing time against energy usage.

The DEFG combines the data that is used for the visualization of the energy flame graphs. Having two main differences, first it uses the data from all available threads to estimate their power consumption and processing time. Furthermore, some added behaviour was added in the case of recursion, a function calling itself.
Recursion can cause large traces, where the same function is called repeatedly. The amount of recursive function invocations is often not relevant, as it often serves as a replacement iterative statements that cause statements to be executed multiple times (for/while loops). In order to keep the data readable, the program will identify direct recursion, a function with an invocation to itself, and attributes them with \((R)\) after the second call. The user can then analyse the recursive functions as a single joined function. An example of this behaviour will be shown and explained in the experiment.

- **CPU Time.** The CPU time is calculated by calculating the total amount of time a stacktrace was active \(((\text{endtime} - \text{starttime}) + 1)\) and subtracting the total time its children (the functions it was calling).

- **Energy Consumption.** The consumption of a particular stacktrace is based on the assumption that the functions within an interval from the energy values are equally distributed. That is, if a consumption of 1000 joules is observed in 1 second, a method that is being processed for 0.4 seconds will, according to the program, account for 40 percent of the consumption and thus consume 400 joules.

First the program will calculate the time-frames in which an element is active on the CPU. An example of this calculation is shown in figure 7.5, which shows the calculation, shown as the red hatched area, of the time-frames for element A. The time that the other elements are active is left out of these timeframes.
7.4 Experimental Setup

The time-frames are then compared to the energy values, based on the percentage of time an element was active in the interval of the energy values is used to determine the joules. Eg, if 50 percent of the interval for the energy data is used by element A, it will be calculated that it accounts for 50 percent of the joules observed during that interval.

The joules consumed are calculated as: “\(\text{watts} \times (TII / DOI)\),” where \(\text{watts}\) is the observed watts consumed, \(TII\) is the amount of milliseconds the time-frame is inside the interval and \(DOI\) the duration of the interval. The calculated joules from all time-frames are added into the total joules consumed.

7.4 Experimental Setup

The energy flame graphs are designed as a proof of concept, as such a simple experiment is used to validate the capabilities. The experiment is performed by a single researcher without any validation, nor multiple experiments to validate the results. Therefore, the data of the experiment cannot be used to say something about the energy consumed of the experimented program.

Instead, this experiment will focus on the usability of energy flame graphs to analyse the energy consumption of an application. Furthermore, potential problems or limitations of the graphs are discussed.

The target system used in the experiments is an ThinkPad Edge E530c notebook, containing of an Intel core I5-2520M CPU with a clock speed of 2.50 GHz and 8 GB...
7. GRAPHICAL REPRESENTATION OF SOFTWARE AND ENERGY

total memory. All unnecessary programs and components were disabled during the collection of the data.

The algorithm used in the experiment is the quicksort algorithm taken from Data Structures & Algorithms in Java [25, pg. 337]. The algorithm was slightly adjusted to only print out 10 numbers instead of the entire array (the numbers are evenly spread throughout the array). The input provided is a randomly generated array of 20,000,000 numbers. The numbers are generated at the start of the experiment and the same numbers are sorted 10 times. A Java-based program monitors the execution of this algorithm and extracts the stacktraces for all active threads in the Java Virtual Machine (JVM).

The energy data is provided by JouleMeter\(^1\) using a new calibration before the start of the experiment. The data is stored into a CSV file, which is picked up by the monitoring program to be stored alongside the stacktrace data.

7.5 Discussion

The entire experiment was performed within 1 minute and 16 seconds. The application spend 9 seconds generating a random list of 20 million numbers and approximately 5.5 seconds for each sorting of the numbers.

First, we can analyse the execution of the application using the visualization of the stacktraces. The processing thread is visualized in figure 7.6 which has the added annotation of the different the sorting algorithms (the runs) that were executed.

The initial visual analysis shows that each of the runs look roughly the same, two main stacks with a small ‘gap’ in the middle. The energy consumption of the experiments further shows similarities between each instance of the sorting function, the energy consumption goes down at the start of the algorithm, followed by a spike of 3-4 watts in consumption.

However, when the individual run is examined more closely, some differences can be seen in the paths. Figure 7.7 shows a closer view of the stacktraces by run 3 and 4. Although both of the runs start roughly the same, a ‘wide’ stacktrace can be seen in the left run which is absent in the right run.

\(^1\)http://research.microsoft.com/en-us/projects/joulemeter/
7.5 Discussion

Figure 7.6: Quicksort Processing Overview - Visualized overview of the processing of a sorting algorithm in stacktraces together with the energy consumption readings. Each complete processing of the sorting function is annotated with the unique identifier “run” 1 to 10.

Figure 7.7: Quicksort Process - A close-up of the stacktraces of two instances of the quicksort algorithm, with the energy consumption in green.
7. GRAPHICAL REPRESENTATION OF SOFTWARE AND ENERGY

The differences are very interesting as the input data is equal across both executions. Such identifications in the code could show the effect of caching functions, bugs in the code causing probabilistic errors. Within the experiment we suspect this to be due to the monitoring program that is capturing the stacktraces, which does not take into account the processing speed of the CPU and therefore can register wrong time values for a captured trace or skip it completely.

Although the data is not completely accurate, the visualization does show useful information about the program execution. Improved profilers with the ability to capture the stacktraces more accurately can further improve such analysis on a lower granularity.

The overview tab of the program shows the Differential Energy Flame Graph (DEFG) for the experiment. The DEFG is shown in figure 7.8 and shows that only a few functions made a clear large contribution to the processing of the application. An important thing to note at this figure, is that the individual elements, as previously described, are not merged.

![Differential Energy Flame Graph](image)

**Figure 7.8: Differential Energy Flame Graph** - The overview of a differential energy flame graph that shows the processing time of each stacktrace (top) and the estimated energy consumption in joules (bottom). The stacktraces themselves are shown in both the top as the bottom view.

As an effect, the view does not show that “Thread.run()” is the function which causes most of the high spikes. Instead the view only shows the influence a specific stack has
on the execution time and energy consumption of the application. One specific stack is noticeable in both the processing time as the energy consumption.

The selected function can be enlarged to show a better view of the results. Figure 7.9 provides a closer view of the function, which is annotated with a 1.

![Figure 7.9: Function Energy and Time Consumption](image)

The figure shows an enlarged view on two methods, recQuickSort (1) and partitionIt (2), for their processing time and energy consumption.

As the view is enlarged, the effects of the surrounding functions becomes more visible, which shows differences in the width of the corresponding components in the top and bottom view. According to data, the recursive invocation of function “recQuickSort” (1) can be attributed to 48.39 percent of the processing time, and only 44.63 percent of the total consumed joules.

The enlarged view of figure 7.9 additionally shows function 2, a partition function in the sorting algorithm is using a lot more processing time as compared to the energy consumption. The view states that “partitionIt” (2) is executed for 3.24 percent of the total processing time, but only consumed 0.29 percent of the energy.

The ability of generating DEFG from experimental data can help developers to find functions that require a lot of time or energy to process. The identification of these functions can attribute to the improvement of efficiency in code. Furthermore, as both the processing time and the energy is compared to each-other, we can discover which functions are heavily influenced by other components. The discovery of non-proportional consumption can help developers understand the energy usage in their applications.
7.6 Threats to Validity

The experiment performed has several threats against its validity, or on the validity of the proof of concept program. The issues identified that pose a threat to its validity are listed in the following paragraph.

- **Result Validation.** The experiment is done by a single researcher and without any validation. The results of the experiment could be influenced by the actions of said researcher. However, as the focus is on the usability instead of the results, we consider this a minor threat.

- **Stacktrace Precision.** The stacktraces used in the program are taken using a monitoring program in Java, which uses functions provided by the Java programming language to extract the stacktraces. These function do not allow to schedule the extraction of stacktracing, and more advanced profiling applications are required to gain more accurate data.

- **Energy Precision.** The energy consumption is monitored using JouleMeter, which bases it on the drainage of the battery power of a mobile device. The results taken from this program might be different from the actual values, and thus provide inaccurate readings.

- **Calculation Errors.** The calculations done within the program are not validated for their accuracy. Incorrect calculation of values can pose a threat to the information gained during analysis of an application.
Part III

Conclusion
Conclusions

8.1 Contribution

The role of software has been acknowledged as one of the factors that contribute to energy consumption in ICT. However, the exact influence that software has on the energy consumption is yet undetermined and is the subject of various research. The objective of this thesis is to support future research into this area.

For this goal, the thesis centres itself around information gathered via experiments which show the effect of software on energy consumption, in order to improve the sharing and understanding of the gathered data. The research is organized in two parts, both answering a part of the main research question. Each question will be addressed individually in the following sections, before answering the main research question.

8.1.1 RQ 1: "What is a suitable data model to represent software and energy consumption?"

Various researchers are investigating the role of software in energy consumption. However, the results from a preliminary literature study showed the irregularity in the data used to measure the energy consumption, effectively preventing the reuse of old valuable data. We presented the benefit of using a common data model to reuse the data taken from previous experiments.

Based on the literature, several requirements were defined for the selection of a suitable model. An exploratory literature review revealed no models satisfying these requirements, instead the search was redirected to models intersecting with individual
8. CONCLUSIONS

areas of the resulting mode. Using the requirements, a semantic data model using the Web Ontology Language (OWL) was created: the Software Energy Consumption Model. The model is designed to be usable without outside influences, that is, other models are not required to fully understand the concepts defined in the model, instead the other models were addressed for possible overlap and connections.

The usability of the model is shown with a small experiment, demonstrating its ability to store the static information on software, its environment, and the data relating to the energy consumption. Although the experiment was conducted in a contained context, without any practical application, we argue the added value of the said model in sharing and storing information.

We present the Software Energy Consumption Model as a suitable data model for representing both software and energy consumption due to its ability to, not only store the relevant information, but also for the capability of defining the relationship between software and energy. Due to its semantic nature, we can analyse the data and known relations using tools to infer and discover new relationships.

8.1.2 RQ 2: “How to visualize the most relevant entities and relationships of the data model for software energy efficiency?”

Aside from the collection of data, the visualization of such data can provide significant contribution in understanding the energy consumption of software. We investigated current techniques used in visualizing software and energy, before providing further analysis on two methods for visualization: a data analysis tool and a novel graphical representation, each being validated using a small experiment to demonstrate the usability.

The selected tool SynerScope allows for visual analysis of datasets. Using the dataset taken from an existing experiment, we showed how the combination of different views can be used to investigate the dataset. The analyses uncovered relationships in resource usage by the experimented applications by showing patterns that influence the energy consumption.

Alongside the analysis tool, we analysed the usage of Flame Graphs in conjunction with data relating to energy consumption. We presented Energy Flame Graphs and Differential Energy Flame Graphs, two experimental implementations of the flame graphs and demonstrated their usability in a proof-of-concept experiment. Even though the
experiment is limited in terms of precision, initial results suggest that the concept can help understanding the relationship between an application and the energy it consumed.

The combination of both methods can be used to visualize the entities and relationships between software and the consumption of energy. For instance, SynerScope can be used for analysis of the entire dataset identifying the relationships within the data, whereas the Energy Flame Graphs provide direct information about relevant locations within an application that consume a lot of energy.

8.1.3 The Main Research Question

The main research question that forms the basis of this work is: “How can data relating to software and the consumption of energy be represented and visualized for analysing energy efficiency?”

The first part of the question is addressed by the sub questions, namely we have presented the definition of a model that can represent data relating to software and energy, and we presented two methods for visualizing this data. The second part relates to the usage of the model and visualization methods for analysing energy efficiency.

Before we can state an answer on analysing energy efficiency, a definition of “efficiency” is required. The Oxford dictionary defines efficient as: “achieving maximum productivity with minimum wasted effort or expense.” From this we state that efficiency defines a comparison, i.e. “being more efficient compared to a counterpart.”

As a result, by itself the “energy efficiency” of software is meaningless. However, even a comparison of multiple software applications required a common context, a shared ‘productivity’ factor that is available in both. A good example for this is the comparison between different application versions, a newer version might be more “energy efficient” when its consuming less energy for the same functionality.

The research described in this thesis provide tools for analysing the Energy Efficiency of software, according to the defined definition for “efficiency”, by providing methods for representing and visualizing the energy consumption of software. The results of these methods can be used to compare and analyse the energy efficiency between different software applications within a given context.

8. CONCLUSIONS

8.2 Future Work

Although the current research has come to a conclusion with this report, there is plenty of work left to be done. The goal of the thesis was to provide innovative methods in supporting research into the energy efficiency of software, acknowledging the amount of research left to be done.

The methods presented should provide a helpful step for the community in understanding software energy efficiency, but require future research into improving their precision.
Appendix A

Data Model

<xml version="1.0"?>
<!DOCTYPE rdf :RDF [
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY dc "http://purl.org/dc/elements/1.1/" >
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY secm "http://few.cs.vu.nl/s2group/ontology/secm#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >]

<rdf:RDF xmlns="http://few.cs.vu.nl/s2group/ontology/secm#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:secm="http://few.cs.vu.nl/s2group/ontology/secm#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:secm="http://few.cs.vu.nl/s2group/ontology/secm#">
<owl:Ontology rdf:about="http://few.cs.vu.nl/s2group/ontology/secm">
<dc:description xml:lang="en">The Software Energy Consumption Model provides the basic entities for analysing the energy consumption of software applications.</dc:description>
<owl:versionIRI rdf:resource="http://few.cs.vu.nl/s2group/ontology/secm_v1.0/"
</owl:Ontology>
</rdf:RDF>
A. DATA MODEL

```xml
<rdfs:range rdf:resource="&secm;Module"/>
<rdfs:domain rdf:resource="&secm;Program_Construct"/>
<owl:inverseOf rdf:resource="&secm;is_constructed_from"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;contains_function">
<rdfs:domain rdf:resource="&secm;Execution_Unit"/>
<rdfs:range rdf:resource="&secm;Function"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;depends_on_workload">
<rdfs:domain rdf:resource="&secm;Execution"/>
<rdfs:range rdf:resource="&secm;Execution_Workload"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;described_by">
<rdfs:domain rdf:resource="&secm;Measure"/>
<rdfs:range rdf:resource="&secm;Measurement"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;describes">
<rdfs:domain rdf:resource="&secm;Measure"/>
<rdfs:range rdf:resource="&secm;Measurement"/>
<owl:inverseOf rdf:resource="&secm;described_by"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;has_environment">
<rdfs:domain rdf:resource="&secm;Execution"/>
<rdfs:range rdf:resource="&secm;Environment"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;has_execution">
<rdfs:domain rdf:resource="&secm;Software_Application"/>
<rdfs:range rdf:resource="&secm;Execution"/>
<owl:inverseOf rdf:resource="&secm;is_execution_of"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;has_hardware_environment">
<rdfs:domain rdf:resource="&secm;Hardware_Resource"/>
<rdfs:range rdf:resource="&secm;Hardware_Environment"/>
<owl:inverseOf rdf:resource="&secm;has_hardware_resources"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;has_hardware_resources">
<rdfs:domain rdf:resource="&secm;Hardware_Environment"/>
<rdfs:range rdf:resource="&secm;Hardware_Resource"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;has_measurements">
<rdfs:domain rdf:resource="&secm;Resource"/>
<rdfs:range rdf:resource="&secm;Measurement"/>
<owl:inverseOf rdf:resource="&secm;taken_from_resource"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;has_software_environment">
<rdfs:domain rdf:resource="&secm;Software_Resource"/>
<rdfs:range rdf:resource="&secm;Software_Environment"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;has_software_resources">
<rdfs:domain rdf:resource="&secm;Software_Environment"/>
<rdfs:range rdf:resource="&secm;Software_Resource"/>
</owl:ObjectProperty>
```
<owl:ObjectProperty rdf:about="&secm;is_execution_of">
  <rdfs:domain rdf:resource="&secm;Execution"/>
  <rdfs:range rdf:resource="&secm;Software_Application"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm;part_of_application">
  <rdfs:domain rdf:resource="&secm;Module"/>
  <rdfs:range rdf:resource="&secm;Software_Application"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm; taken_during_execution">
  <rdfs:domain rdf:resource="&secm;Measurement"/>
  <rdfs:range rdf:resource="&secm;Execution"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm; taken_from_resource">
  <rdfs:domain rdf:resource="&secm;Resource"/>
  <rdfs:range rdf:resource="&secm;Measurement"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm; used_by_function">
  <rdfs:domain rdf:resource="&secm;Function"/>
  <rdfs:range rdf:resource="&secm;Function"/>
  <owl:inverseOf rdf:resource="&secm; uses_function"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm; used_in">
  <rdfs:domain rdf:resource="&secm;Environment"/>
  <rdfs:range rdf:resource="&secm;Execution"/>
  <owl:inverseOf rdf:resource="&secm; has_environment"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm; used_in_unit">
  <rdfs:domain rdf:resource="&secm;Function"/>
  <rdfs:range rdf:resource="&secm;Execution_Unit"/>
  <owl:inverseOf rdf:resource="&secm; contains_function"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="&secm; uses_function">
  <rdfs:domain rdf:resource="&secm;Function"/>
  <rdfs:range rdf:resource="&secm;Function"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="&secm;data_is_of_type">
  <rdfs:domain rdf:resource="&secm;Execution_Workload"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&secm;has_data">
  <rdfs:domain rdf:resource="&secm;Execution_Workload"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&secm;has_duration">
  <rdfs:domain rdf:resource="&secm;Execution"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&secm;has_name">
  <rdfs:domain rdf:resource="&secm;Measure"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&secm;has_property">
  <rdfs:domain rdf:resource="&secm;Resource"/>
</owl:DatatypeProperty>
A. DATA MODEL

```xml
<owl:DatatypeProperty rdf:about="&secm;has_timestamp">
  <rdfs:domain rdf:resource="&secm;Execution"/>
  <rdfs:range rdf:resource="&xsd;date_time"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&secm;has_unit">
  <rdfs:domain rdf:resource="&secm;Measure"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&secm;has_value">
  <rdfs:domain rdf:resource="&secm;Measurement"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&secm;has_version">
  <rdfs:domain rdf:resource="&secm;Module"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="&secm;Environment">
  <rdfs:label xml:lang="en">Environment</rdfs:label>
  <rdfs:comment xml:lang="en">The environment is a generalized concept used to describe a collection of resources in which an application can be executed. The entity is strictly used as an abstract definition which is further defined in software and hardware environment concepts.</rdfs:comment>
</owl:Class>

<owl:Class rdf:about="&secm;Execution">
  <rdfs:label xml:lang="en">Execution</rdfs:label>
  <rdfs:comment xml:lang="en">The &apos; execution &apos; entity represents data related to the finished execution of a software application. The entity is the collection of the workload, application, and environment and links this to the collected measurements and runtime attributes. Using the execution multiple different test runs of an experiment can be identified and distinguished.</rdfs:comment>
</owl:Class>

<owl:Class rdf:about="&secm;Execution_Unit">
  <rdfs:label xml:lang="en">Execution Unit</rdfs:label>
  <rdfs:comment xml:lang="en">The execution unit is a logical container within the source code in which functions can be defined. Depending on the underlying language the container can be classes, packages, or files. Providing an intermediate layer between the functions and modules in which they are defined.</rdfs:comment>
</owl:Class>

<owl:Class rdf:about="&secm;Execution_Workload">
  <rdfs:label xml:lang="en">Execution Workload</rdfs:label>
  <rdfs:comment xml:lang="en">The data provided by an application can provide an unique execution of a program. As the a specific set of instruction can lead distinct
```

The execution of a program, making it important data when an execution is analysed.

The workload entity allows for reuse in multiple experiments and identifying the executions run based on a specific workload.

Although data properties are currently added to the function entity it can be extended with additional program metrics if there is need for these metrics.

Note that the environment is specific to a single configuration. Meaning that when the configuration changes the environment would also be different, making sure there is a distinction between the environment when used in different experiments.

The hardware resources can be used to model different hardware components, physical or virtual, such as CPU or memory and connect the components to an hardware environment.

This means that the entity is used for dynamic, time dependant, data about a resource within a certain execution.
defining them explicitly. In turn, reused modules can be used in future studies to investigate the comparative consumption and efficiency compared to different versions or similar modules.

The hardware environment is managed by an operating system, which typically links to the software environment. As the operating system can have a significant impact on the performance of an execution and therefore qualifies to be defined as an entity itself.

Some methods for the estimation of software energy consumption analyse properties of the program. The program construct is an abstraction for the construction elements that make up a module, such as the programming language(s) and functions.

A library is a pre-defined set of instructions that can be imported and used by an application to perform a certain function. The workings of a library cannot be directly changed by the actual program itself (with the exception of behaviour that can be influenced without adjusting the source). Examples: JSON Parsing, JDBC, Log4J.

There is a wide selection of programming languages that can be used to write software, not only with syntax but also with the underlying library. The differences between these languages are often not easily distinguishable. As such the modelling of the language can become a valuable asset. Not only can default or additional language-specific information be added, with enough data the effects between languages can be studied.

Similar to the environment, the resource entity is a generalized concept to describe a resource within an environment. The underlying entities further define the resource by the separation into hardware and software.
A software resource is a resource that contributes to the execution of a program that is defined as a 'software' component.

For example: Java Virtual Machine, Common Language Runtime, Operating System.

Listing A.1: The Software Energy Consumption Model in OWL.
A. DATA MODEL
References


REFERENCES


