Behavior-Based Consumption Profiles for the Approximation of the Energy Consumption of Services

Jorge Andrés Larracoechea
LIUPPA / E2S – Université de Pau et des Pays de l'Adour
Anglet, France *jorge-andres.larracoechea@etud.univ-pau.fr*

Philippe Roose
LIUPPA / E2S – Université de Pau et des Pays de l'Adour
Anglet, France *Philippe.Roose@iutbayonne.univ-pau.fr*

Sergio Ilarri
Instituto de Investigación en Ingeniería de Aragón / I3A – Universidad de Zaragoza
Zaragoza, Spain *silarri@unizar.es*

Yudith Cardinale
Dpto. de Computación y T.I – Universidad Simón Bolívar
Caracas, Venezuela *ycardinale@usb.ve*

Sébastien Laborie
LIUPPA / E2S – Université de Pau et des Pays de l'Adour
Anglet, France *sebastien.laborie@iutbayonne.univ-pau.fr*

Olga Paulina Vara
Universidad Popular Autónoma del Estado de Puebla
Puebla, México *olgapaulina.vara@upaep.edu.mx*

Abstract

Regardless of the improvements in the efficiency of energy consumption of information and communication technology, energy consumption will forever be a requisite for software execution. Consequently, researchers have promoted the development of green and sustainable software with new development methods and tools. These, however, have been adopted with limited success due to technicalities and specific language/platform requirements. In this paper we introduce a portion of our Behavior-Based Consumption Profiles (BBCP). A platform and language agnostic software behavior profiling approach, aimed at estimating the energy consumption of software from the analysis and design phases of the Software Development Life Cycle (SDLC). The profiles, in a JSON format, contain properties whose values provide and control specific descriptions of the software's behavior. Throughout the paper, these properties and their underlying mechanics are explained from a perspective of software services, to conclude with an experiment where a real-world service is profiled and its BBCP is simulated to obtain its behavior.

Keywords: SOA, Green Software, Sustainability, Software Profiling, Software Engineering

1. Introduction

Even if the optimization of information and communication technology (ICT) has resulted in an increasingly frugal software execution, software will always be responsible for the energy expended to achieve a computational result. Due to it, the researchers in the field of ICT have segmented the construction of frugal software into a new branch of studies named Green Soft-

ware Development (GSD). In a previous contribution [9], we provided a brief literature review of the existing tools and methods for the management and assessment of the energy consumption of software, from a perspective of SOA (service-oriented architectures) and taking GSD into account. In the aforementioned contribution, we concluded that the existing approaches back, overall, the development and maintenance stages of the Software Development Life Cycle (SDLC), leaving software architects and designers unsupported during the initial stages of it.

This conclusion motivated us to extend our contribution with further research on the approaches available for software behavior modelling, making a strong emphasis on GSD and focusing on service-like units of software. The conclusions of this extended study are available in [8], and they are the following: (1) most of the existing tools and methods available cater to specific platforms and frameworks, making their adoption cumbersome. (2) There is a research gap between the initial stages (analysis and design) of the SDLC and the later stages of it, as none of the approaches exist within a complete GSD methodology that bridges them. (3) Despite a previous proof of unawareness [12], the audience targeted by these approaches is increasingly aware of software's energy consumption [5], but the adoption of the existing tools and methods is hindered by their steep learning curve. In the same paper we proposed a set of properties, most of them qualitative, that constitute what we named a Behavior-Based Consumption Profile (BBCP). The purpose of the BBCP is to describe the amount of hardware resources a unit of software consumes, based on how and when the software is used.

We concluded the paper by recognizing several challenges, some of them being: (1) a logic for interpreting the properties of the BBCP and generating a behavior with them, (2) successfully approximating a resource consumption from the behavior and (3) the development of a tool that facilitates the creation and interpretation of BBCPs, so that a representative resource consumption can be translated into a final energy consumption per profile.

In the spirit of solving the first challenge, the objectives of this paper are the following: (1) introduce and define the first parts of the BBCP and (2) explain how they work together to provide a behavior. Due to the limited extent of this paper, we have restrained the content covering the first objective to only 15 out of 43 properties. The properties presented in this paper define a temporal frame and how it evolves, as it establishes when a behavior (or lack of) occurs. Without the features presented here, our approach would lack a temporal plane where hardware consumption can be evaluated. Throughout the paper, we will look at software from a perspective of services: a unit of software responsible for a business function [4].

In Section 2 we discuss the existing approaches dedicated to the description of software's behavior, in Section 3 we introduce a set of the elements of our BBCP approach as well as their mechanics. In Section 4 we create a profile after a real-world example of a service, we assess its behavior in Section 5, and discuss the integration of our BBCP approach into green software development methodologies in Section 6, closing the paper with a discussion on future work and our final conclusions in Section 7.

2. Related Work

Studies on the behavior of software and the behavior with software are not new in the field of computer science. On one hand, several examples of the latter include taking into account human personality profiles to generate believable bot behaviors [14], prototyping intelligent systems that adapt their behavior to match the profile of the user [1] and the simulation of real user behavior on the internet using machine learning techniques [10]. On the other hand, several examples of the former include studying the behavior of component-based software in embedded systems according to different signal flows [6], characterizing profiles with the behavior of an application and its resources in heterogeneous environments [11] and the creation of behavioral profiles built on UML (Unified Modeling Language) that specify the behavioral rules of their architecture [7].

Most of these works make use of a common technique: profiling. Profiling allows researchers to study a specific part of a subject by taking its descriptive elements and characterizing a profile with them. This, in turn, allows them to deepen their knowledge of the subject without dealing with its full complexity. For instance, in [11] application's behavioral profiles are characterized by a task processing capacity in quantifiable dimensions, such as the RAM and the storage usage. In [7], the application's behavioral profile takes place in the form of UML stereotypes that allow the authors to trace the application's interactions in an environment. Other approaches aim at modeling the behavior of services with their own language, aimed specifically at behavioral dependencies [17].

These works, however, fail to address some aspects that concern the assessment of energy consumption from a model of the software's behavior. Even though the execution traces and signal flows can be taken into account for the behavior of software components, a global and granular view of the system is required to model the impact that the evolution of the behavior can cause on its energy consumption. Sequence diagrams are helpful for understanding the behavior of an application and its interactions with the surrounding elements, but they do not provide a description of a diachronic behavior that includes elements of uncertainty, such as Human-Provided Services [15], where people can be actors that affect a system's behavior. Finally, we could not find any software behavior modeling language that interprets behavior as a source of energy consumption.

In addition to the previous observations, we defined several needs that our BBCP approach has to address to fully emulate the behavior of a service and how its environment affects it: (1) define the intensity of the usage of a service (how much the service is used) and include a controlled variability, representative of external factors such as HPS or business related constraints. (2) Constrain the usage of a service to account for and prevent edge cases, such as its over and under use. (3) Define multiple time frames when the behavior of the service should shift. (4) Determine how the behavior mutates during specific time frames, to study how the diachronic behavior of services affects their energy usage. (5) Have a declaration of the operations that the service executes, what they represent and the requirements that need to be met before they take place. (6) The operations declared must also provide a declaration of the hardware they consume, so that operations can be identified as a source of energy consumption. (7) Be able to declare triggers, inputs and outputs per operation, so that the dependencies among them can be managed as execution paths, and later studied as energy relationships where data can play a mayor role.

The fulfilment of these needs is crucial, as it will allow software designers and system architects to develop an expectation of how an application and its parts will behave from the conception phase, and independently of the type of service (or any other unit of software). The approximation of the energy consumption will give them the give designers and architects a better understanding of how the software should be shaped in order to reduce their impact on the consumption of resources and, therefore, the energy consumption. In a direct relation to these needs, we have concluded that our profile should be characterized by the requirements in Table 1.

Now that the requirements for our profile have been established, we will continue the paper in the following Section 3 presenting the first 15 properties of our BBCP, as well as the mechanics they contribute with in compliance to the requirements specified above.

3. Elements of the Behavior Based Consumption Profile

The Behavior-Based Consumption Profile is, simply put, a collection of properties with individual roles and rules, whose purpose is emulating the behavior of software and how its environment affects its behavior. These properties, are distributed in 7 main categories that match the requirements explained in Section 2. Due to the restricted length of this contribution, we will

Table 1. Requirements taken into account to characterize our profiles

Requirement	Name	Description
1	Usage	Provide a mechanic that defines isochronal points in time when the use of a service can be decided, and add a stochastic approach to it that provokes uncertainty, so that external factors such as Human-Provided Services (HPS)[15] can be taken into account.
2	Run constraints	Provide constraints that enforce run time policies, affecting the future usage of a profile.
3	Expectations	Provide a format that defines expectations and collections of time frames at different scopes, such as days and hours.
4	Variability	Provide sets of values that, during valid expectations, override or mutate the values responsible for controlling the behavior of the service.
5	Operations	Provide a solution for declaring as many operations as needed within the profile, as well as the requirements needed to execute them.
6	Hardware consumption	Allow for a quantitative representation of hardware consumption per operation.
7	Dependency management	Allow the definition of elements within the profile that express parameter I/O, operation triggers and relationships among operations

limit this section to the explanation of the properties in categories 1 through 3. As we previously mentioned, the remaining properties that address requirements 4 through 7 will be introduced in a future contribution. To ease the process of locating each of the properties, a table with relational graphs that explain the order (place within the BBCP) is available in Figure 4 (Appendix A), while Table 3 (Appendix A) provides the value type, format and possible values for each property.

3.1. Requirement 1: Usage

The first category, *usage*, groups properties that describe the most basic behavior that the profile will exhibit: the shift between its main states, *run* and *stop*. The *run* state represents the active use of the profile and the *stop* state the opposite. The shift between these two states is managed with *evaluations*. Evaluations are isochronal points (points at regular intervals) in time when the shift is decided. They are defined with the property *Profile Evaluation Rate* (PER), (row 1 of Table 3). Its value, in Hz, provides the frequency at which evaluations are spread throughout an indefinite amount of time.

Take, as an example, a service that recollects the video feed of a camera each 10 minutes. A *PER* representative of this basic behavior would have a value equal to 0.016 Hz. As stated in Table 1, this category is not complete without including a stochastic solution to model uncertainty. *Run base probability* (row 2 of Table 3) creates with its value a simple variation in the behavior of the profile. During an evaluation, the chance for the profile to enter a run state is evaluated against this probability if and only if the profile is in a stop state. *Stop base probability* (row 3 of Table 3) represents a specific probability for the profile to switch to a *stop* state when the profile is evaluated under a *run* state. Continuing the previous example, a movement sensor could be responsible for triggering the recollection of the camera feed. To model this external uncertainty, we can set a *PER = 1 Hz*, a *run base probability = 0.3* representative of a sensor activation, and a *stop base probability = 0.5* representative of a lack of sensor activation.

Even though these properties begin to build a behavior that meets the needs of requirement 1, edge cases ant time related constraints still need to be taken into account, as we described in requirement 2.

3.2. Requirement 2: Run Constraints

The properties in this category is enforce time policies by limiting the time a profile spends in a *run* state. They are located within the *run constraints* category, or row 2 of Figure 4. In

difference to the previous properties, the restrictions set by run constraints are independent of evaluations. The first property, *minimum run time* (row 4 of Table 3), locks the profile in a run state until its duration reaches the value of this property.

Maximum run time (row 5 of Table 3) limits the maximum duration per run to its value. With it, the profile is switched immediately to a stop state when the duration of a single run surpasses the maximum run time. Going back to the video feed recollection service example, we could set a *minimum run time = 300 seconds* and a *maximum run time = 600 seconds*. In this way, the video recollection would be active for at least 5 minutes and a maximum of 10 continuous minutes at most per run. In this example, we have constrained the simplest edge cases to prevent the under and over use of a profile. There are, however, other use cases that need to be taken into account to allow other time-related constraints that alter the behavior to be expressed.

For instance, a service could have a limited quota of time that it can spend in an active state throughout a period of time. The property *quota* (row 6 of Table 3) is responsible for it, as it limits the total accumulated run time of a profile, independently of how many times the profile enters a run state. Once a profile's accumulated run time reaches this value, it is immediately switched to a stop state and the profile's evaluation is suspended. We call this mechanic a "depleted quota". In our example, we could set a quota for video feed recollection equal to 7200 seconds. Therefore, the profile could spend a total of 7200 accumulated seconds of run time before depleting its quota. In order to replenish a depleted quota, the *Cooldown* (row 7 of Table 3) property defines the total time the profile should suspend its evaluations. To illustrate this pair of properties, let us consider a video streaming service inside of an application of the like of Netflix. As an example, a free user account could be assigned a quota of 3600 seconds for streaming and a cooldown of 3600 seconds. As we can see, the flexibility of the properties in this category allows us to consider functional and non-functional requirements alike.

Now that we have explained how the properties above restrict the run time of a profile, further detail should be employed to fully satisfy requirement 3, as time frames constraint the feasibility of every mechanic described so far.

3.3. Requirement 3: Expectations

The properties in this category are responsible for *expectations*. Expectations are collections of *time frames* and *time frames* are bounded periods of time when specific shifts in behavior occur. In the BBCP, 2 different definitions of time frames exist: *cycles* and *events* (rows 3 through 5 of Figure 4). *Cycles*, as the name states, are responsible for dictating isochronal cyclic shifts in the behavior through time. For example, a cycle could shift the behavior of a profile every weekend for each week. *Events*, however, are single unrepeatable shifts in the behavior; for example, a specific date in time. The multiplicity of expectations allows multiple cycles and events to exist per profile. The contents of cycles and events continue in the following sub-sections.

Cycles

There are 2 properties that complete the definition of a cycle: *scale* and *days*. *Scale* (row 10 of Table 3) is responsible for setting a bound of days within cycles can be defined. For example, setting the scale equal to 7 would imply that time for the cycle is counted in sets of 7 days. The *Days* (row 9 of Table 3) property is used to precise a selection or range of days of the scale when the cycle is valid. Revisiting our streaming service example, if we set a scale equal to 7, one selection of days could represent the weekends: [6-7] and another selection the rest of the week: [1-5]. Therefore, we could precise a general shift in behavior depending on the day of the week.

Events

Events consist of three basic properties: *scale*, *initial day* and *ending day*. The *Scale* property (row 10 of Table 3) sets an absolute reference of time for the event to take place. For example, setting the scale to 365 would define a year of reference for events to take place. The *Initial day* (row 11 of Table 3) property defines a specific number of day in which the time frame for the event is started. The *ending day*, as the name implies, sets the number of day in which the event is finished. Going back to our example, events could mark specific shifts in the behavior of the streaming service during new content release. For instance, if we consider a scale equal to 30, we could have 2 different events for content release, one with an initial date on day 14 and ending date on day 15, and another one with an initial date on day 29 and an ending date on day 30.

Even though at this point we are able to create bounds of days, we still need to refine our time frames by creating a selection at our second level of granularity: *hours*. In order to achieve this, we distributed these properties into their own instanceable sub-category: *timed expectations*, as seen in row 5 of Figure 4.

Timed Expectations

Timed expectations define a shift in the profile's behavior in an hourly basis, as we previously defined in requirement 3. The multiplicity of this set of properties allows us do define several instances of it per event and per cycle. The *time range* property (row 13 of Table 3) allows the bounding of time within a specific selection of single hours or ranges of hours. Once a time range has been selected, the first property to shift the behavior of the profile during it is the *profile evaluation rate override* (similar to row 1 of Table 3). This property overrides with its value the original *PER*, increasing or decreasing the frequency of the evaluations. The *run probability override* property (row 14 of Table 3) overrides the profile's run base probability with its value, while the *stop probability override* (row 15 of Table 3) does the same to the profile's stop base probability.

In the following section, we will present an example of a BBCP we created to showcase how these and the previous properties we explained aggregate to a final representation of behavior.

4. BBCP Example

In order to provide a useful example of how each set of properties of the BBCP translate into a specific behavior, we decided to profile a service that plays a major role in a Video Game Streaming Application (VGSA). A VGSA is a video game execution application consisting of a cloud computing architecture. In it, a constant exchange of data between the cloud and the client takes place. The cloud (a computer cluster in the network) receives peripheral data input from the client and performs graphics processing tasks with it. The cloud, in turn, provides the client with the video and audio data that the client uses in turn as an input. This sequence is perpetuated in a constant feedback loop until the user quits the application.

We chose this application due to its increased adoption and its reliance on human behavior, as big technology and software development companies such as Google (Stadia), Nvidia (GeForce Now) and Sony (PS Now) compete to put their infrastructures at the consumers' disposal to through their own VGSA, lifting the hardware requirements from the consumers. Even though this application can be decomposed into several services, the service we will profile is the game stream service, responsible for the feedback loop between the client and the cloud after a game is selected off a catalogue of available games. Furthermore, we will profile the constrained usage that a “free" user (a trial account) would generate on the service by using run constraints.

The data fragments required to characterize our profile are the following: (1) a video frame

rate, (2) time restrictions based on the type of user and, due to the nature of these applications where the usage is dictated by the users, (3) data on the service usage throughout the day.

The video frame rate relies greatly on the available network bandwidth the user has at his disposal [2]. We selected a value equal to 60 frames per second in order to model ideal network conditions on the user's side, but the profile consist of a variable PER to simulate a spotty network. This was achieved by overriding the PER at different time ranges. For the second data fragment, time restrictions, we chose a quota equal to 1 hour. This limit was selected after the restriction of the "free" user account in Nvidia's VGSA, GeForce Now. We added a cooldown equal to 1 hour to deliberately constraint the profile further, and observe how the mechanic restricts the total run time. For the final data fragment, service usage throughout time, we consulted the total active users throughout a week's day and a weekend's day for the most played game at the time. This information was obtained from Steam Charts [16], a website that analyses the usage data of the most popular online digital games distribution platform, Steam, owned by Valve. The plots of both of the samples are available in Figure 5 and Figure 6, where, the X axis represents the change in time and the Y axis the amount of active concurrent players.

These 3 basic fragments of data allowed us to charaterize a BBCP, available in its JSON format below:


```
{"Usage": {
          "profile evaluation rate": "60",
          "run base probability": "0.5",
          "stop base probability": "0.5",
          "Run constraints": {
            "minimum run time": "600",
            "maximum run time":"3600",
            "quota": "3600",
            "cooldown":"3600"
        }},
  "Expectations": {
        "Cycles": [{
            "scale":"7",
            "days":"[1-5]",
            "Timed expectations": [{"time range": "[0-8]",
                "profile evaluation rate override":"60",
                "run probability override": "0.1",
                "stop probability override": "0.9"
              },{"time range": "[8-16]",
                "profile evaluation rate override":"45",
                "run probability override": "0.3",
                "stop probability override": "0.7"},{
                "time range": "[16-23]",
                "profile evaluation rate override":"30",
                "run probability override": "0.9",
                "stop probability override": "0.1"}]},{
            "scale":"7",
            "days":"[6-7]",
            "Timed expectations": [{"time range": "[0-8]",
                "profile evaluation rate override":"60",
                "run probability override": "0.3",
                "stop probability override": "0.7"
              },{"time range": "[8-12]",
                "profile evaluation rate override":"50",
                "run probability override": "0.5",
                "stop probability override": "0.3"},
                {"time range": "[12-23]",
                "profile evaluation rate override":"45",
```

```
                "run probability override": "0.9",
                "stop probability override": "0.1"}]}]}
}
```

5. Results

We created a system model constituted by the properties and mechanics presented during section 3. The objective of this experiment was to prove that the sample of the properties of the BBCP presented in this paper can aggregate to an expected behavior constrained by well defined frames of time. The simulator we used to model our system is Insight Maker, a "general-purpose web-based simulation and modeling tool" [3]. We chose Insight Maker due to its low learning curve, high flexibility, collaborative features, support of system dynamics and, last but not least, free and open source nature. The resulting model for our experiment is accessible through the following URL: `https://insightmaker.com/edit/3Rf21SBOHcAntcT3nPojDa/access`. Our model was fed with the JSON format of the profile in section 4. We decided to scale down the PER of our example to 1/100 of the original value to maintain a good performance during the simulation. After making sure that the simulation executed correctly, we sampled a day of the first cycle.

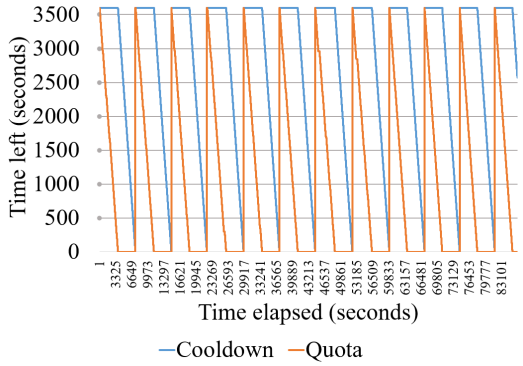

Fig. 1. A chart of the quota and cooldown constraints' values over time

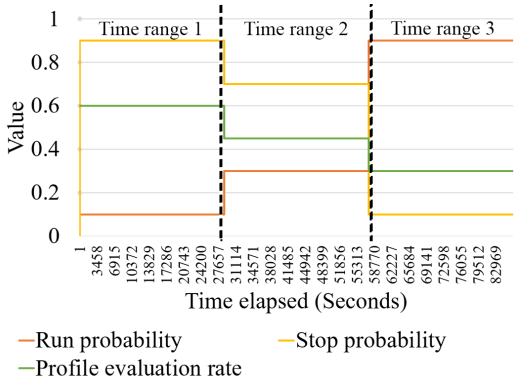

Fig. 2. A chart of timed expectations entering into effect at their specific time ranges

The analysis of the behavior in this sample can be divided into 3 main observations: respected constraints, matched expectations, and final run time obtained. In Figure 1, we can observe how the quota and cooldown behaviors match the current run time of Figure 7. Furthermore, the run time is never less than 600 seconds long unless the quota is depleted, or more than 3600 seconds (per run instance or accumulated run). The constrains are, therefore, respected with success. In Figure 2, evidence for the success of the profile's expectations is observed. The main state probabilities as well as the profile's PER are altered through time, in accordance to the time range of each timed expectation of the profile. Finally, an accumulated run time equal to 43200 seconds is obtained.

5.1. Brief Discussion

As it can be seen in the **current run time** of Figure 7, the state of the profile is constantly switched without spending a significant amount of time in a stop state, resulting in a high accumulated run time. We are aware that this behavior juxtaposes the profile's frame rate with the necessity of emulating the users' behavior as a HPS. This paradigm is solved with a "shell" profile: a profile that provides a complementary behavior to a rich (completely modeled) profile. A shell profile with a less **intense** PER (high frequency over time) could trigger the profile in

section 4, increasing the fidelity of the behavior. The elements responsible for the solution of this paradigm will be disseminated in a future contribution.

6. A Design Methodology with BBCP

With the example above, we demonstrate that characterizing the properties presented here requires a reduced amount of knowledge on the behavior of the user and the behavior of the service, as well as optional business rules that can reflect non-functional requirements that affect the consumption of a service. We envision the BBCP as an approach that, in the near future, could be incorporated during the analysis and conception phases of the SDLC.

Further development of our approach could provide full-stack green software development methodologies such as the one proposed in[13] with continuous support along the later stages of the SDLC, where BBCPs generated along the analysis and design phases are carried on and re-assessed at each step of the SDLC, as seen in Figure 3. This, in turn, will evolve our BBCP approach into a exchangeable format of behavior and categorization of energy consumption that enables systems to adapt to frugal configurations preemptively in self-adaptive systems.

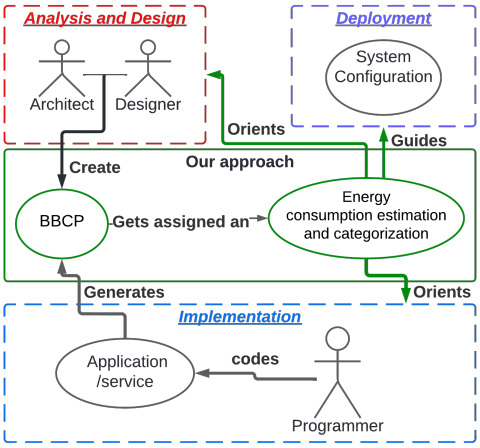

Fig. 3. The model of the inclusion of our approach within the SDLC

7. Conclusions and Future Work

In this paper, we presented 15 out of the 43 properties that constitute a Behavior-Based Consumption Profile. We achieved this by explaining each of their characteristics and mechanics, as well as their categorical distribution. We also provided an example where we analyzed and profiled a specific service from a real world video game streaming application, showcasing our approach and obtaining an expected behavior with it. Even though we believe that our approach is a step towards sustainable software and systems design, as well as the democratization of GSD, there are several research directions that are still missing to incorporate it into a green software development methodology.

The profiling process described above is too conceptual and cumbersome to be adopted without a tool that assists our future users in the process of characterizing a BBCP. In addition to this, the remaining 28 properties responsible for the requirements 4 through 7 and a final estimation of energy consumption still require dissemination among the scientific community. Finally, our approach as a whole still requires validation. We will address, in a future contribution, these research directions towards a cohesive support for green software design and development along the SDLC.

Acknowledgements

This research has been supported by the project PID2020-113037RB-I00/AEI/10.13039/501100-011033 and the Government of Aragon (Group Reference T64_20R, COSMOS research group).

References

1. Benton, S., Altemeyer, B., Manning, B.: Behavioural Prototyping: Making Interactive and Intelligent Systems Meaningful for the User. In: 2010 International Conference on Intelligent Networking and Collaborative Systems. pp. 319–322 (Nov 2010). https://doi.org/10.1109/INCOS.2010.51
2. Di Domenico, A., Perna, G., Trevisan, M., Vassio, L., Giordano, D.: A network analysis on cloud gaming: Stadia, GeForce Now and PSNow. Network **1**(3), 247–260 (Oct 2021). https://doi.org/10.3390/network1030015, http://arxiv.org/abs/2012.06774, arXiv: 2012.06774
3. Fortmann-Roe, S.: Insight Maker: A general-purpose tool for web-based modeling & simulation. Simulation Modelling Practice and Theory **47**, 28–45 (Sep 2014). https://doi.org/10.1016/j.simpat.2014.03.013
4. IBM: SOA vs. Microservices: What's the Difference? https://www.ibm.com/cloud/blog/soa-vs-microservices (May 2021), accessed: 2022-04-17
5. Jagroep, E., Broekman, J., van der Werf, J.M.E.M., Brinkkemper, S., Lago, P., Blom, L., van Vliet, R.: Awakening awareness on energy consumption in software engineering. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Society Track. pp. 76–85. ICSE-SEIS '17, IEEE Press, Buenos Aires, Argentina (May 2017). https://doi.org/10.1109/ICSE-SEIS.2017.10, https://doi.org/10.1109/ICSE-SEIS.2017.10
6. Kim, J.E., Kapoor, R., Herrmann, M., Haerdtlein, J., Grzeschniok, F., Lutz, P.: Software Behavior Description of Real-Time Embedded Systems in Component Based Software Development. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). pp. 307–311 (May 2008). https://doi.org/10.1109/ISORC.2008.69
7. Koskinen, J., Kettunen, M., Systä, T.: Behavioral profiles-a way to model and validate program behavior. Software: Practice and Experience **40**(8), 701–733 (May 2010). https://doi.org/10.1002/spe.977
8. Larracoechea, J., Roose, P., Ilarri, S., Cardinale, Y., Laborie, S., González, M.: Towards Services Profiling for Energy Management in Service-oriented Architectures. In: 17th International Conference on Web Information Systems and Technologies. pp. 209–216 (Jan 2022)
9. Larracoechea, J.A., Roose, P., Illari, S., Laborie, S., Cardinale, y., Gonzalez, M.J.: Modeling Energy Consumption in SOA: Requirements and Current Status. In: Atelier "Évolution Des SI" (INFORSID 2021). Dijon, France (Jun 2021)
10. Niu, X., Liu, H., Xin, G., Huang, J., Li, B.: User Application Behavior Sequence Generation. In: 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA). pp. 466–469 (Aug 2020). https://doi.org/10.1109/AEECA49918.2020.9213508
11. Östberg, P.O.: A Model for Simulation of Application and Resource Behavior in Heterogeneous Distributed Computing Environments. In: SIMULTECH. pp. 144–151 (2012)
12. Pang, C., Hindle, A., Adams, B., Hassan, A.E.: What Do Programmers Know about Software Energy Consumption? IEEE Software **33**, 1–1 (Jan 2015).

https://doi.org/10.1109/MS.2015.83
13. Roose, P., Sergio, I., Larracoechea, J.A., Cardinale, Y., Laborie, S.: Towards an integrated full-stack green software development methodology. In: 29th International Conference on Information Systems Development (Sep 2021)
14. Rosenthal, C., Congdon, C.B.: Personality profiles for generating believable bot behaviors. In: 2012 IEEE Conference on Computational Intelligence and Games (CIG). pp. 124–131 (Sep 2012). https://doi.org/10.1109/CIG.2012.6374147
15. Schall, D., Truong, H.L., Dustdar, S.: Unifying Human and Software Services in Web-Scale Collaborations. Internet Computing, IEEE **12**, 62–68 (Jun 2008). https://doi.org/10.1109/MIC.2008.66
16. SteamCharts: Steam charts - counter-stike global offensive. https://steamcharts.com/app/730#48h (2022), accessed: 2022-04-15
17. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.: Let's Dance: A Language for Service Behavior Modeling. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. pp. 145–162. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11914853_10

Appendix A

	Order	Properties
1	Usage	Profile Evaluation Rate Run Base Probability Stop Base Probability
2	Usage — Run constraints	Minimum run time Maximum run time Quota Cooldown Countdown
3	Expectations — Cycles	Days Scale
4	Expectations — Events	Initial day End day Scale
5	Expectations: Cycles — Timed expectations; Events — Timed expectations	Time range Run probability override Stop probability override Run constraint (an instance from row 2)

Fig. 4. The BBCP properties and their order

Table 2. The most valuable concepts we elaborated on throughout Section 3

Name	Description
Behavior	Explains when and how the service uses resources.
Time frame	A bounded period of time where actions are constrained; a temporal description of behavior.
Expectations	Collections of predefined time frames.
Timed expectations	A collection of time frames bounded by hours.
Cycles	A collection of cyclic time frames bounded by days, with a variable scale and instances of timed expectations and run constraints.
Events	A collection of well bounded, non cyclic time frames bounded by days; with instances of timed expectations.

Table 3. The BBCP properties and their constraints

Row	Name	Numeric	Range	Possible values
1	Profile Evaluation Rate	x		$x \geq 0$ in Hz, if $x = 0$ the property is ignored.
2	Run Base Probability	x		$0 \leq X \leq 1$
3	Stop Base Probability	x		$0 \leq X \leq 1$
4	Minimum run time	x		$x \geq 0$ in seconds, if $x = 0$ the property is ignored.
5	Maximum run time	x		$x \geq 0$ in seconds, if $x = 0$ the property is ignored.
6	Quota	x		$x \geq 0$ in seconds, if $x = 0$ the property is ignored.
7	Cooldown	x		$x \geq 0$ in seconds, if $x = 0$ the property is ignored.
8	Countdown	x		$x \geq 0$ in seconds, if $x = 0$ the property is ignored.
9	Days		x	Ranges: [1-7] Selections and ranges: [1-3,6]
10	Scale	x		$x > 0$ in days.
11	Initial day	x		$x \geq 0$ if $x = 0$ the property is ignored.
12	End day	x		$x \geq 0$ if $x = 0$ the property is ignored.
13	Time range		x	Ranges: [1-12] Selections and ranges: [1-12,17]
14	Run probability override	x		$0 \leq X \leq 1$
15	Stop probability override	x		$0 \leq X \leq 1$

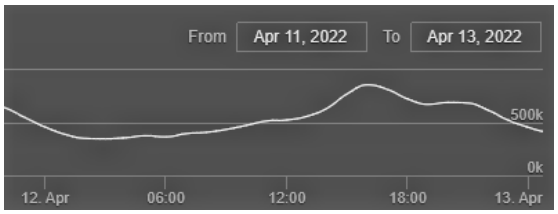

Fig. 5. Week's day concurrent users of the top game at the time

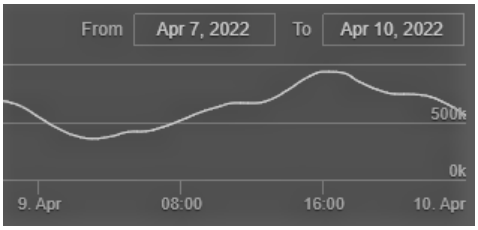

Fig. 6. Weekend's day concurrent users of the top game at the time

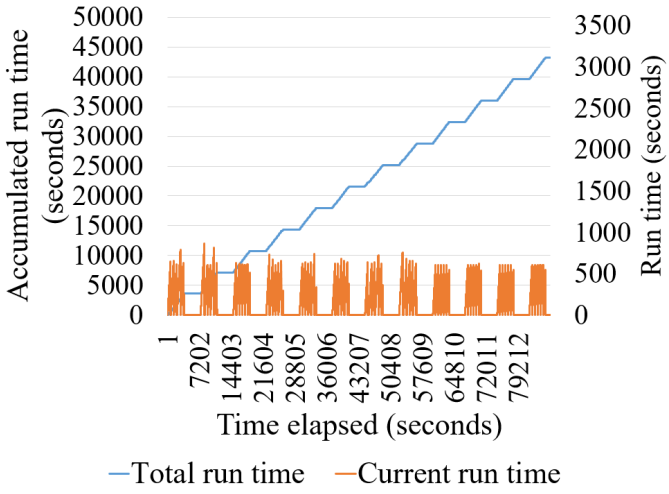

Fig. 7. Chart of the total time the profile spent in a run state, as well as per time per run