Green Framework Patterns - Strategies for Embedding Sustainability into Application Frameworks

Balawal Sultan, Free University of Bozen-Bolzano
Jacopo Ammendola, Free University of Bozen-Bolzano
Eduardo Guerra, Free University of Bozen-Bolzano

While many companies today pursue sustainability, there are not many initiatives related to making software more sustainable, also known as "green IN software", reducing waste of computational resources. In this work, we point to frameworks as an important point to introduce sustainable solutions while they can potentially impact several applications. We describe four patterns, namely Built-in Sustainability in Frameworks, Sustainable Adapter, Sustainable Configuration, and Sustainability Panel, that aim to capture practices that can make them suitable for building more sustainable applications.

Categories and Subject Descriptors: Software and its engineering [**Software organization and properties**]: Software system structures—*Software architectures*; Software and its engineering [**Software creation and management**]: Designing software—*Software design engineering*; Information systems [**Data management systems**]: Information integration—*Mediators and data integration*

General Terms: Software Architecture

Additional Key Words and Phrases: green software, sustainability, software design, framework, patterns

ACM Reference Format:

Sultan, B. Ammendola, J. Guerra, E. 2024. Green Framework Patterns - Strategies for Embedding Sustainability into Application Frameworks HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 22 (October 2015), 12 pages.

1. INTRODUCTION

As digital technologies evolve, their environmental impact has become a critical concern. The energy consumption and carbon footprint associated with software applications [Eva Kern 2013] can be small when we look individually at one device, but when accumulated across thousands of devices, it becomes significant. For that reason, it is important to integrate sustainability into the development process [Thibault Simon 2023] and consider it an important quality attribute in the architecture. In this paper, we document patterns that can be used by software applications to reduce the waste of computational resources, in particular, through the usage of frameworks.

When a framework wastes computational resources, it can affect many applications that implement it [Calero et al. 2021]. However, what is a threat can also be an opportunity since sustainable solutions can also be embedded into development frameworks. This way, the usage of sustainable solutions can be transparent to developers, not adding significant development costs.

These patterns also bring an idea that creates a new dimension to save resources: empowering users to make sustainable choices with the application usage. Since users can have different profiles, an essential feature for one can be a waste of resources for another. Allowing users to configure the application to promote sustainable behavior according to their profile can reduce waste.

The adopted pattern format begins with a brief context, highlighting in which cases the guideline should be applied. Next, the problem statement summarizes the sustainability issue addressed by the pattern, supported by a list of forces contributing to the problem. Then, the solution section presents the proposed approach for adding sustainability into frameworks, providing steps for implementation within software systems. After, a concrete example of the pattern usage is presented in italics, followed by a list of consequences.

In the following sections, we will explore four patterns that aim to achieve the goal of making software applications more environmentally sustainable: EMBEDDED SUSTAINABILITY IN FRAMEWORKS, SUSTAINABLE ADAPTER, SUSTAINABLE CONFIGURATION, and SUSTAINABILITY PANEL. More in detail, we will explore why these patterns are needed, what the possible challenges are in implementing them, some example use cases, and the consequences of using these patterns.

2. BUILT-IN SUSTAINABILITY IN FRAMEWORKS

AKA Embed Sustainability in Frameworks, Sustainability Behind the Curtains

Implementing sustainable solutions into software systems can be complex and require additional development time. When prioritizing the development team's work, functional features or other quality attributes are usually prioritized instead of sustainability. In applications that use frameworks, most developers are not aware of how sustainable the approach used internally is. When a framework is used by several applications and does not provide sustainable solutions, the impact that can be small for one application might be substantial when multiplied by several users across several systems.

How to use sustainable solutions without increasing the application development cost?

—**Complexity** : Sustainable solutions often require significant changes to standard development practices, introducing complexity that usually demands more effort.

—**Resource and Time Constraints** : Development projects are typically bound by tight schedules and limited resources, making the additional effort required for sustainable practices challenging to justify.

—**Lack of Awareness and Tools** : There's a gap in awareness among developers [Torre et al. 2017] about the availability and benefits of sustainable practices, compounded by a scarcity of tools to facilitate their implementation.

—**Enough Performance** : When a software system has a performance that is enough to fulfill the requirements, the additional effort required to reduce the waste of resources and make the solution more sustainable is not prioritized.

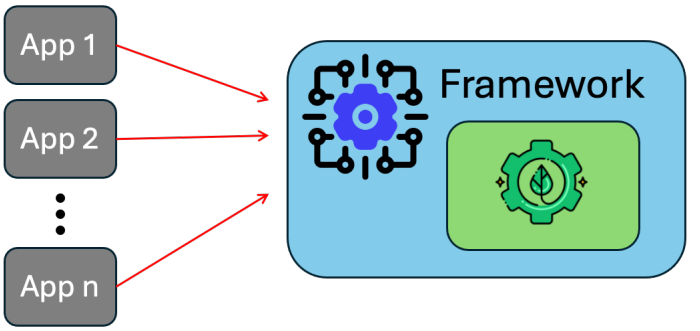
—**Not used features** : Some frameworks provide features that, even when they are not used, can add processing load.

—**Different requirements** : Applications have different requirements and priorities for quality attributes, such as reliability and security.

—**Repeated waste** : A small waste of resources in a framework functionality repeated several times in several applications by several users can result in a big waste.

Therefore:

Embed sustainable solutions directly inside frameworks so applications can reuse them without a high impact on development effort.



Since frameworks provide a backbone for building applications, they are perfectly positioned to incorporate these sustainable practices seamlessly. By EMBEDDING SUSTAINABILITY INTO FRAMEWORKS, more sustainable solutions are transparently used by applications that implement that framework. That also ensures that sustainability becomes a standard practice rather than an afterthought [Ardito et al. 2015].

Efficient solutions that avoid the waste of resources can be used internally in the framework. Their complexity will be hidden from the applications and will be easier to be reused. These solutions should seek to use less memory, less processing, and a lower number of IO operations, such as database access and networking. Another possibility to avoid waste is to allow a SUSTAINABLE CONFIGURATION through the parametrization of the framework. This way, it is possible to configure the most sustainable behavior for each scenario.

BUILT-IN SUSTAINABILITY IN FRAMEWORKS also helps to maintain a fast development pace while integrating sustainability without imposing significant additional burdens on developers. Highlighting such sustainable practices in the framework documentation will raise the awareness of sustainability as an important quality attribute among the development community. In case the sustainable solution can be parameterized, that part must be clear in the documentation.

Depending on how the sustainability of the solution is embedded into the framework, developers might find limitations in its impact on other quality attributes and end up choosing a different framework with fewer limitations. Therefore, the framework can also provide other options that can be configured by the application developers[Ardito et al. 2015]. For example, by having a framework that addresses other quality attributes in some scenarios, the more sustainable solution configured by default can be used in others. In that sense, extensibility in these critical points provides more possibilities.

Consider a framework that aims to synchronize a list of items with remote applications. While a solution could be to always send the full list of items to be updated, that solution wastes resources by repeatedly sending the same information. Following this pattern, the framework could internally adopt a more sustainable solution, sending only the changes in the data. While this solution is more complex to implement, it would be inside the framework, and it would not affect the applications that use it.

Consider that a framework consumes data from files and performs some validation of the data format. In a system where this verification is already done previously while the file is generated, executing it again is a waste of resources. If the framework allows this verification to be disabled, this application can use it to avoid the waste of having the verification executed twice. However, if this feature is disabled when loading data from other sources, there is a higher risk of corrupted data being loaded.

The following are the consequences of using this pattern:

—**Sustainability impact (+)** : Considering that the framework can be used by several applications by several users, the impact in reducing waste, even if small individually, can be huge.

—**Easy adoption (+)** : Embedding sustainable practices into the frameworks lowers the barrier to implementing sustainable solutions [Torre et al. 2017] and encourages its widespread adoption.

—**Cost reduction (+)** : Developers do not need to spend additional development time, reducing the respective cost to make the application sustainable.

—**Adaptability (+)** : The framework can be adapted to applications in which sustainability is a priority but also when other quality attributes are more critical.

—**Hidden sustainability (+)** : When the developers use the framework without paying attention to the gains in sustainability, the framework still works in the background to make the application more sustainable.

—**Lack of Awareness (-)** : If developers are not aware of the importance of sustainability as a quality attribute, they might neglect it in other parts of the application.

—**Undesired trade-offs (-)** : The sustainable solution might have trade-offs that sacrifice other quality attributes, such as security or reliability, that the development team might not be aware of.

—**Framework Development Effort (-)** : The effort to implement the sustainable solution in the framework can be significant.

3. SUSTAINABLE ADAPTER

AKA Sustainability in the Middle

Some APIs are not developed prioritizing sustainability as a quality attribute. However, in some cases, the developers do not have control over the API, and in others, they do not want to change it because of the legacy code that already uses it. The API can be part of a well-known and widely used framework, which can have other components that already make use of the API. If the API is used in several parts of the code, changing the way it is used requires significant effort.

How to add sustainability transparently in the usage of an existing API that you do not have control over?

—**Minimizing Changes** : If the application already uses the existing API, it is desirable not to have many changes in the code.

—**Request Optimization** : Requests from the application can be optimized before forwarding them to the API. This optimization may involve aggregating multiple requests into batch requests, compressing data payloads, or caching frequently accessed data to minimize redundant calls.

—**Developer Habit** : The developers are more likely to use an API if they are used to it.

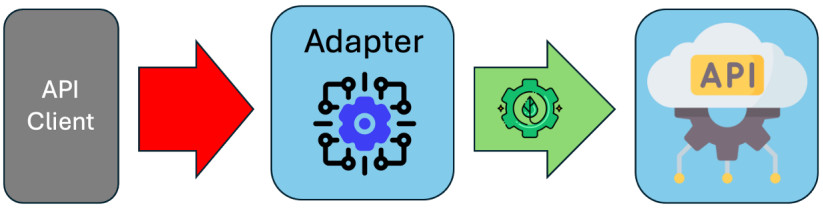
—**Control for Modification** : The development team might not have modification access to the API to add a sustainable solution internally.

—**Multiple API Implementations** : The same API might have several implementations, and optimizing the usage for only one of them does not affect the others.

—**Solution Reusability** : If the application optimizes the usage of an API in its own code, it will not be possible to be reused in other applications.

Therefore:

Create a reusable adapter that wraps the original API and optimizes its usage to save resources.



The core of this pattern is to create an adapter [Gamma et al. 1993; **?**] that wraps existing APIs, like interfaces of development frameworks, external systems, service endpoints, or components, to add more sustainable solutions to software applications. These are some strategies that can be used by the adapter to reduce the consumption of resources: (a) aggregating multiple requests into batch requests can reduce the number of requests using a Request Bundle [Zimmermann et al. 2023], (b) compressing data payloads can reduce the amount of data transferred, and (c) caching frequently accessed data help to minimize redundant calls. By making this adapter independent of the application, it became reusable in other contexts.

A dynamic adapter can use metadata [Guerra et al. 2013] to create a framework that provides a more general solution that can be used for several APIs. That solution can be used in a metadata-based framework that aims to provide this kind of solution. A framework can provide a SUSTAINABLE ADAPTER for its own APIs, allowing framework users to use it or not. In the same way, a framework can also provide SUSTAINABLE ADAPTERS for a specific API and can be reused in any application that uses that API.

This approach allows developers to seamlessly incorporate or detach these adapters, thereby introducing sustainable functionalities as required. The primary advantage of this method is its transparency, which allows one to introduce sustainability features to fit the diverse needs of different applications without requiring significant changes to their fundamental architecture. These adapters can be integrated into reusable frameworks that can add these sustainability features to several applications.

For example, in applications that display data through dynamic tables that have frequent updates, the integration of a caching adapter provided by the framework could significantly reduce the need for constant server requests. This adapter could intelligently cache table data, thereby minimizing energy consumption associated with data fetching operations. Alternatively, the framework could offer an interface that adjusts the frequency of data refresh operations, perhaps by requesting new data only on every other call, effectively halving the energy and bandwidth usage associated with maintaining data currency.

The class GreenRequestManager presented in Listing 1 is an example of the implementation of this pattern to reduce the number of requests to refresh data in JavaScript frameworks. This class wraps the method that requests data from the server and can be configured to reduce the number of requests, minimizing unnecessary network traffic and server load, leading to reduced energy consumption both on the client side and server side. This implementation is general and can be reused for different APIs.

The attribute `greenRequests` stores information about each request, including how many times it has been made (count), and caches the last result received. The other attribute, `greenRequestsRate`, holds the rate at which requests should be repeated. The method `makeGreenRequest()` receives as a parameter the function that performs the server request. So, this function only calls the original one according to the rate configured, reducing the number of requests. Since Javascript frameworks often work with asynchronous calls, `observers` attribute holds the callback functions that should be called when new information comes from the server.

The following are the consequences of using this pattern:

—**Sustainability impact (+)** : The adapter can reduce the waste of resources when used with the API.

—**Code Impact (+)** : The impact of adding the adapter in a code that already uses an existing API is low.

—**Configurability (+)** : Separating the optimization from the core API behavior gives the client application the possibility to use it or not.

—**Separation of Concerns (+)** : The sustainability concern is isolated from the others inside the adapter.

—**Adapter Cost (-)** : The adapter execution also has a cost in terms of memory and processing, and to be worth its usage, the amount of resources that it saves should be higher than the amount it consumes.

—**Implicitness (-)** : Since the usage of the adapter might not be explicit in the code, it might be hard to know if it is active or not.

Listing 1. Example of a Sustainability Adapter to reduce the number of repeated requests

```
1
2  class GreenRequestManager {
3      constructor() {
4          this.greenRequests = new Map();
5          this.greenRequestsRate = new Map();
6          this.observers = []; // List of observers to notify
7      }
8      async makeGreenRequest(request) {
9          const requestName = request.name;
10         let result = undefined;
11         if (!this.greenRequests.has(requestName)) {
12             result = await request();
13             this.greenRequests.set(requestName, { count: 0, lastResult: result });
14             this.greenRequestsRate.set(requestName, 1);
15         } else {
16             const requestObject = this.greenRequests.get(requestName);
17             requestObject.count++;
18             if ((requestObject.count % this.greenRequestsRate.get(requestName)) === 0)
                   {
19                 requestObject.lastResult = await request();
20                 this.greenRequests.set(requestName, requestObject);
21             }
22             result = requestObject.lastResult;
23         }
24         this.notifyObservers();
25         return result
26     }
27     setGreenRequestsRate(requestName, rate) {
28         this.greenRequestsRate.set(requestName, rate);
29         this.notifyObservers();
30     }
31 }
```

4. SUSTAINABLE CONFIGURATION

AKA Sustainability First, Sustainability Parameters

Many frameworks assume that some quality attributes, like requirements related to security or reliability, are desired by all applications. However, in some cases, the application developers might want to prioritize sustainability as a quality attribute. An important feature for one application might not be needed for another, and the execution of an unneeded one that executes by default can result in a waste of computational resources. The application developer needs to accept the provided behavior if the framework does not provide an API to change parameters

or customize the processing.

How to avoid waste of resources in frameworks used by applications with different needs?

—**Unused features** : Some frameworks provide several features, and not all of them might be needed by all applications that implement them.

—**Load of unused features** : The presence of features not used depending on the implementation might add processing load even when they are not used.

—**Different requirements** : Applications have different needs and priorities for quality attributes, such as reliability and security.

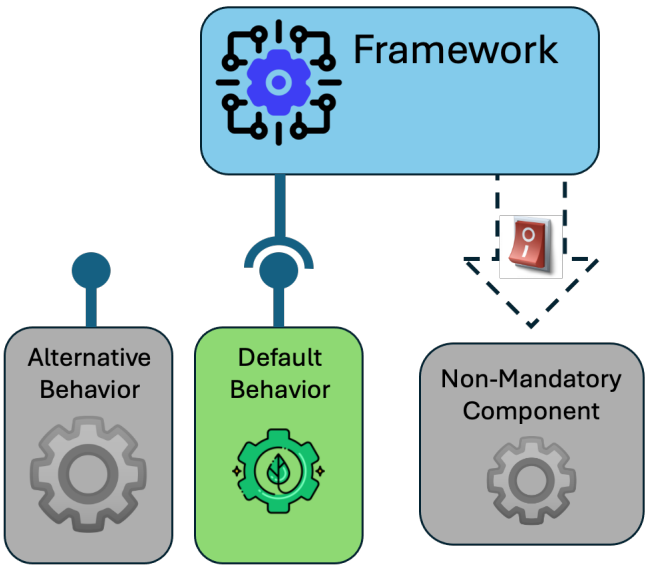
—**Repeated waste** : A small waste of resources when passing through an unused framework functionality repeated several times in several applications can result in a big waste.

—**Default Behavior** : If the most sustainable behavior is not the one used by default, it might not be activated by negligence.

—**Extensibility** : Frameworks are extensible by nature and can provide different behaviors for the same feature.

Therefore:

Provide an API that allows applications to configure framework behavior to have a suitable sustainability trade-off for their needs.



Developers can introduce parameters in the framework to allow the choice of features that prioritize different quality attributes. This configuration can be done, for instance, when instantiating framework classes, in the invocation of framework methods, with configuration files, or by adding class metadata (like Java code annotations) [Guerra 2016]. In cases where different features can have different requirements for the quality attributes, the

framework can adopt a configuration for each functionality instead of a general one for the whole application. When different users have different needs, the configuration can be different for each framework request. To avoid wasting resources by negligence, the most sustainable behavior can be the one configured by default, allowing the other behaviors by configuration.

Choices that directly influence the amount of resources used by the framework, such as thread pool sizes, caching strategies, or frequency of background tasks, are good candidates for configurable parameters. Another point that the framework can provide parameters to configure is granular control over individual features, enabling developers to selectively enable or disable specific functionalities based on their application's requirements and priorities. Examples of these features are validating data received externally and logging the framework activities. When the framework follows the recommendation to provide the most sustainable alternative as default, the developers can activate features or increase the resources used only when needed.

The framework should be designed internally to avoid extra processing for disabled features. As an example, a conditional that verifies the configuration to execute or not that feature or a search for a file needed to execute it are examples of approaches that will be executed and consume resources in every execution. Patterns in which behavior can be plugged or unplugged, such as DECORATOR, PROXY, CHAIN OF RESPONSIBILITY, and COMPOSITE, would be good alternatives, since they are configured just once.

The framework developer should not assume that certain quality attributes are universally desired. When they are not mandatory by that application or by that functionality, it can potentially lead to the waste of computational resources. Those parameters would let application developers prioritize sustainability as a quality attribute and customize framework features accordingly.

Comprehensive documentation and guidance about the existing parameters should accompany the framework, outlining best practices, examples, and recommendations for optimizing resource usage and prioritizing sustainability. The documentation can present a guide to achieving sustainability in the framework, highlighting the trade-offs made by each configuration [Kern et al. 2013].

As an example, consider that a framework consumes data from files and performs some validation of the data format. In a system where this verification is already done previously while the file is generated, executing it again is a waste of resources. By following SUSTAINABLE CONFIGURATION*, the validation would be disabled by default, and the framework would provide an API that allows this verification to be enabled when needed by the application.*

The following are the consequences of using this pattern:

—**Sustainability impact (+)** : This pattern can avoid the execution of unnecessary framework features and the allocation of unnecessary resources.

—**Flexibility (+)** : The framework can be adapted to the needs of several applications.

—**Modular Structure (+)** : By allowing features to be disabled, they will become decoupled from the core framework features, driving their design towards a modular structure.

—**Internal Complexity (-)** : Introducing parametrization to disable features can make the framework's internal structure more complex.

—**Configuration API Complexity (-)** : Configuring the framework parameters can bring additional complexity to the application.

—**Learning Curve (-)** : Disabling not mandatory features might work against having a GENTLE LEARNING CURVE[Foote and Yoder 1996].

5. SUSTAINABILITY PANEL

AKA Empower Users to be Sustainable

In modern software applications, user needs and preferences can vary widely, including their requirements for how frequently data should be updated or refreshed within an application. These preferences can significantly impact the sustainability of an application, as more frequent updates can lead to higher energy consumption and increased server load. In some cases, different users might have different needs that might require a different consumption of resources. For example, a user might need to refresh the data on the screen at a different frequency than others. In the same way, even the same user might have different needs at different moments.

Since the applications usually do not let users change their parameters, it is not possible for them to configure how the system works if they want to make a more sustainable choice. Another thing to consider is that most end users of software applications have no real way of understanding the environmental impact of their use of the application.

How to allow application users to make sustainable choices based on the usage profile?

—**Lack of Understanding** : Most end users have no understanding of the environmental impact of using software applications.

—**Lack of Visibility** : Most applications do not provide visibility to the end users of their usage of resources.

—**Impact on Sustainability** : More frequent data updates or refreshes can lead to higher energy consumption and increased server load, impacting the overall sustainability of the application.

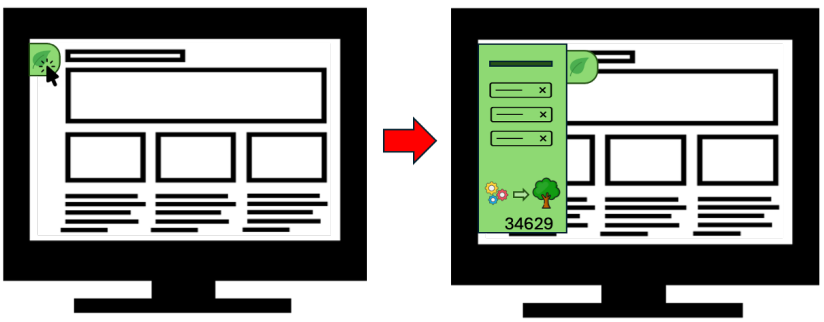
—**Unnecessary Feature** : Executing features that are not necessary to all users, like a sidebar with a chart that shows live information, can waste computational resources.

—**User Profile** : Depending on the user profile, the need for other quality attributes, such as accuracy or performance, might differ. Even the same user might have different needs in different situations.

—**User Choice** : Some users might choose to favor sustainability over optimal performance.

Therefore:

Provide a configuration panel where the users can see and change options that affect sustainability, having a vision of the resources that can be saved and the impact on the application usage.



The Sustainability Panel serves as a user interface component that provides visibility into the environmental impact of the application's usage patterns and allows users to customize their interactions to align with sustainability goals. It allows the user to have a more sustainable choice in how the application is used. This panel should provide a suitable user interface that considers the expected user knowledge about software applications and sustainability.

The first step is to have SUSTAINABLE CONFIGURATION in the framework and identify which parameters would make sense for the user to configure. Examples of such parameters could be related to parameters of AI algorithms [Van Wynsberghe 2021; **?**] and data refresh rates. Other possibilities are to allow users to disable widgets and features that provide additional information, such as showing hints for filling out forms.

The panel can also display data on the environmental impact of the user's interactions with the application. This may include metrics such as energy consumption and carbon footprint [au2 et al. 2024] or other relevant sustainability indicators. Comparisons with daily life, like with the energy from a lamp, can also help the users to understand how much they are saving. In cases where the energy saving by an individual is not so much, but the impact considering all users is huge, this information can also be shown. By visualizing this information, users gain awareness of the environmental consequences of their actions within the application.

Frameworks can have a sustainability panel associated with them. This will allow the same panel to be reused in several applications. If more than one framework shares the same API for configuring the parameters, the same panel can be reused for all of them. The panel can be initially hidden on the page, with a small button or icon in the corner that can be used to open it. Having the same panel approach used in several applications can create familiarity in users, who can recognize and use it more easily.

Considering the code example of the GREENREQUESTMANAGER *adapter, a panel could present users with options to adjust how often the application refreshes table data, ranging from real-time updates to more energy-conserving intervals. For example, users could opt for a lower refresh rate, thereby conserving bandwidth and reducing the application's overall energy consumption.*

Listing 2 presents a code snippet from the GreenWidget class, which defines a sustainability panel for the adapter presented on Listing1. Users can interact with the GreenRequestManager and alter how often the requests managed by the GreenRequestManager are forwarded to the server. The method UPDATEREQUESTRATE() *is the one called to update the request rate based on user input. It calls the setGreenRequestsRate() method and sets the rate configured by the user. This panel can be reused on any page in which the adapter encapsulates any function that makes requests to the server.*

The following are the consequences of using this pattern:

—**Empowered users (+)** : Users can choose to be more sustainable in using the software according to their usage profile.

—**Improved user engagement (+)** : By showing the user how many resources are saved through their choices in the panel, users can feel like a part of the change.

—**More environmentally conscious users (+)** : The sustainability panel can help educate end-users to make more sustainable choices [Torre et al. 2017], not just in software, but in everyday life.

—**Potential Risk of Developer Complacency (-)** : A notable concern with embedding sustainability features directly into applications, particularly through user-configurable panels, is the potential for developer complacency. This might manifest as a reliance on end-users to activate and manage these sustainability options themselves rather than integrating optimized, sustainable practices at the core of the application's development [Pérez et al. 2023].

—**Unsuitable configurations (-)** : Users might configure the parameters with unsuitable values to their profile, and the application might not present the behavior they expect [Ardito et al. 2015].

—**Panel Resources Cost (-)** : The panel is a functionality that consumes resources itself, so the savings should compensate for that cost.

Listing 2. Code Example for the GreenWidget

```
class GreenWidget {
    //ommited methods
    updateRequestRate(event) {
        const requestName = event.target.name;
        const newRate = parseFloat(event.target.value);
        this.manager.setGreenRequestsRate(requestName, newRate);
    }
    render() {
        let menuOpenedText = this.menuOpened ? 'Close Menu' : 'Open Menu';
        let menuOpenedStyle = this.menuOpened ? '' : '"display:none"';

        this.container.innerHTML = `
            <div class="widget">
                <button class="open-menu-btn">
                    ${menuOpenedText}
                </button>
                <div class="side-menu" style=${menuOpenedStyle}>
                    <ul>
                        ${Array.from(this.manager.greenRequestsRate).map(([request,
                            rate]) => `
                            <li>
                                ${request}, call function every:
                                <input name="${request}" type="number" value="${rate}"
                                    />
                                occurrances
                            </li>
                        `).join('')}
                    </ul>
                </div>
            </div>
        `;

        // Assign event listeners after rendering
        this.container.querySelector('.open-menu-btn').addEventListener('click',
            this.toggleMenu.bind(this));
        Array.from(this.container.querySelectorAll('input[type=number]')).forEach(input
             => {
            input.addEventListener('change', this.updateRequestRate.bind(this));
        });
    }
}
```

6. CONCLUSION

In conclusion, the patterns discussed in this paper present practical strategies for integrating sustainability into frameworks. By incorporating sustainability within development frameworks, offering configurable options for sustainability, and introducing user-centric sustainability panels, developers have the potential to address critical challenges concerning environmental impact and user preferences. These patterns serve as proactive measures aimed at reducing energy consumption, optimizing resource utilization, and fostering more sustainable software practices.

However, their effective implementation depends heavily on collaborative efforts among developers, users, and stakeholders to prioritize sustainability and execute these patterns successfully.

It is necessary to recognize the importance of future research in this area. Despite the advancements, there remains a noticeable gap in current research surrounding sustainability in software development. Further investigation and experimentation are necessary to refine and expand the proposed patterns. This ongoing research is crucial for cultivating a greener and more environmentally conscious future within the tech industry. By continuing to explore innovative solutions and strategies, we can address emerging challenges and advance the adoption of sustainable practices across the software development landscape.

ACKNOWLEDGMENTS

Special thanks to Andreas Fießer for their invaluable guidance and contributions, which significantly enriched the evolution of this article. We also thank all the members of our writer's workshop, especially Joseph Yoder, Hugo Sereno, Cesare Pautasso, Daniel Pinho, Zishan Rahman, and James Episale.

REFERENCES

Luca Ardito, Giuseppe Procaccianti, Marco Torchiano, and Antonio Vetro. 2015. Understanding Green Software Development: A Conceptual Framework. *IT PROFESSIONAL* 17 (01 2015), 44–50. DOI:http://dx.doi.org/10.1109/MITP.2015.16

Iztok Fister Jr. au2, Dušan Fister, Vili Podgorelec, and Iztok Fister. 2024. Profiling the carbon footprint of performance bugs. (2024).

Coral Calero, Macario Polo, and Mª Ángeles Moraga. 2021. Investigating the impact on execution time and energy consumption of developing with Spring. *Sustainable Computing: Informatics and Systems* 32 (2021), 100603.

Stefan Naumann Achim Guldner Timo Johann Eva Kern, Markus Dick. 2013. Green Software and Green Software Engineering – Definitions, Measurements, and Quality Aspects. In *ICT4S 2013, Proceedings of the First International Conference on Information and Communication Technologies for Sustainability, ETH Zurich, February 14-16, 2013*, Vol. A4. 87.

Brian Foote and Joseph Yoder. 1996. The selfish class. In *Third Conference on Patterns Languages of Programs (PLoP'96) Monticello, Illinois*.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*. Springer, 406–431.

Eduardo Guerra. 2016. Design patterns for annotation-based apis. In *Proceedings of the 11th Latin-American Conference on Pattern Languages of Programming, SugarLoafPLoP*, Vol. 16. 9.

Eduardo Guerra, Eduardo Buarque, Clovis Fernandes, and Fábio Silveira. 2013. A flexible model for crosscutting metadata-based frameworks. In *Computational Science and Its Applications–ICCSA 2013: 13th International Conference, Ho Chi Minh City, Vietnam, June 24-27, 2013, Proceedings, Part II 13*. Springer, 391–407.

Eva Kern, Markus Dick, Stefan Naumann, Achim Guldner, and Timo Johann. 2013. Green software and green software engineering - definitions, measurements, and quality aspects.

Carlos Pérez, Ana C. Marcén, Javier Verón, and Carlos Cetina. 2023. A Survey on Green Computing in Video Games: The Dawn of Green Video Games. (2023).

Romain Rouvoy Joël Penhoat Thibault Simon, Pierre Rust. 2023. Uncovering the Environmental Impact of Software Life Cycle. *International Conference on Information and Communications Technology for Sustainability, Jun 2023, Rennes, France. ffhal-04082263f* (June 2023).

Damiano Torre, Giuseppe Procaccianti, Davide Fucci, Sonja Lutovac, and Giuseppe Scanniello. 2017. On the Presence of Green and Sustainable Software Engineering in Higher Education Curricula. (2017).

Aimee Van Wynsberghe. 2021. Sustainable AI: AI for sustainability and the sustainability of AI. *AI and Ethics* 1, 3 (2021), 213–218.

Olaf Zimmermann, Mirko Stocker, Daniel Lübke, Uwe Zdun, and Cesare Pautasso. 2023. *Patterns for API Design - simplifying integration with loosely coupled message exchanges* (1 ed.). Vol. 1. Pearson Addison-Wesley.

Received May 2015; revised September 2015; accepted February 2015