An Enhanced Formalism for Resource Management Policies Specification and Fast Evaluation in Pervasive Systems

David Beserra and Jean Araujo

Abstract Pervasive systems demand flexible, efficient resource management policies to handle heterogeneous infrastructures and varying application needs. This paper introduces an extended formalism that overcomes limitations in previous approaches by distinctly separating static properties from dynamic context elements, allowing more precise policy definitions. Mandatory and optional policies are explicitly categorized, enabling fail-fast decisions when critical conditions fail, while also supporting opportunistic executions. These design choices reduce evaluation costs—often down to O(1) in the best case—and permit large-scale environments to benefit from parallel evaluations. Practical simulations demonstrate superior performance in collaborative, multi-organization scenarios, highlighting improved adaptability, reduced overhead, and effective integration of organizational knowledge within the resource management process.

1 Introduction

Pervasive systems are designed to seamlessly integrate a wide range of services and infrastructures, enabling intelligent environments capable of dynamic adaptation [14]. These systems are characterized by their operation across heterogeneous, distributed, and often resource-constrained environments, where computational resources such as CPU, memory, and network bandwidth are shared among diverse tasks [8] [1]. Effective management of these resources is crucial to ensure both the performance of individual tasks and the overall system stability.

David Beserra
École Pour l'Informatique et les Techniques Avancées (EPITA), Paris, France
e-mail: david.beserra@epita.fr

Jean Araujo
Universidade de Aveiro, Aveiro, Portugal
e-mail: jean.araujo@ua.pt

Resource allocation in pervasive systems poses several challenges due to the inherent variability of the infrastructure and the context-dependent nature of service requirements [2]. Tasks often have unique conditions for execution, necessitating well-defined policies that govern resource utilization. These policies must address both static resource properties, such as hardware specifications, and dynamic contextual conditions, such as current system load or network latency.

Existing approaches for police formalization in resource management for pervasive systems and related technologies are limited by hardcoded policies [7], inefficient sequential evaluations, lack of prioritization for mandatory conditions, and poor adaptability to dynamic resource attributes. This paper addresses these limitations by introducing a formalism for resource management policies in pervasive systems. Building upon a previously established model [3][2], which categorized policies into **Requeriments** (must-have conditions) and **Restrictions** (must-not conditions), the extended formalism incorporates several innovations. It introduces hierarchical policy categorization, distinguishing between **Mandatory** and **Opportunistic** policies to enable explicit prioritization. A fail-fast mechanism halts evaluations early when invalid conditions are encountered, reducing computational overhead.

To demonstrate the effectiveness of the proposed formalism, this paper presents a practical example of its application in a collaborative resource-sharing scenario involving multiple enterprises. The scenario illustrates how the formalism supports fairness, efficiency, and adaptability in dynamic environments. Furthermore, a comparative analysis evaluates the computational complexity of the extended formalism and its predecessor, supported by performance simulations to validate its benefits.

2 Related Works

Numerous studies have explored context-aware resource management in pervasive, IoT, edge, and fog computing environments. Examples include [7], [4], [11], [13], [6], [11], and [3][2]. In [7], a centralized, context-aware scheduler for pervasive grids targeting MapReduce applications is proposed, while [4] describes hierarchical strategies for edge/fog environments. Studies [10] and [11] focus on a distributed IoT/edge scheduler in a smart building, and [12] employs edge computing to execute opportunistic services. In [6], the authors analyze various uses of context information for scheduling in edge/fog and IoT environments. Further, [1] addresses resource management in 5G/6G networks, and [3][2] present a fully distributed resource manager emphasizing opportunistic usage.

Despite these contributions, most of these works provide only partial or ad-hoc policy specification approaches, often intermingling policy definitions directly with scheduling or decision logic. First, many fail to clearly distinguish static properties (immutable attributes) from context elements [9] (dynamic, sensor-driven data)—including research such as [6]—thereby complicating policy specification and ignoring differences in data-access costs (rapid for properties yet slower for sensor data). Second, policies are frequently hardcoded into decision mechanisms, limiting flexibility when accuracy requirements or organizational contexts change. Third, most approaches (except [3][2]) overlook organizational aspects, such as cross-company resource sharing. Lastly, none explicitly differentiate mandatory from opportunistic policies, missing opportunities to ensure quality of service while facilitating opportunistic execution.

Study [3] introduced a formalism that partially separates context elements from static properties within a fully distributed resource manager architecture for pervasive systems, thereby enabling more precise resource evaluations. However, it still offers only limited means of structuring and categorizing policies, leaving gaps in how mandatory or optional requirements are formalized. Although [12] supports distributed scheduling and opportunistic resource utilization, it does not explicitly distinguish between mandatory and optional policies, nor does it fully leverage organizational knowledge or variable data accuracy. Thus, while these approaches mitigate some policy hardcoding, important gaps persist in policy specification and formalization.

In this work, we extend [3][2] by introducing a more robust formalism for policy definition. We incorporate mandatory policies (to ensure QoS and accelerate decision-making upon violations) alongside optional policies (to support opportunistic usage). We also propose more structured categorizations—including the separation of static properties versus dynamic context elements, plus consideration of organizational knowledge—to enhance clarity, adaptability, and overall performance in policy specification and evaluation.

3 Management Policies Formalism

This section introduces the formalism for defining and evaluating resource management policies in pervasive systems. Policies serve two main purposes: defining conditions under which a service can be executed on a device and specifying the conditions for a device to accept service execution. This dual-purpose approach ensures comprehensive management of interactions between services and resources in dynamic, heterogeneous environments. Key concepts—properties, context elements, remarkable states, and policies—are formally defined and illustrated to aid understanding.

3.1 Original Formalism

The original formalism presented in [3] [2] provides a structured framework for specifying resource management policies. Policies govern conditions for resource allocation and service execution, ensuring precise control over dynamic interactions between services and devices.

Policies are divided into two categories: **Requeriments** (conditions that should be true for a policy to be satisfied) and **Restrictions** (conditions that should be false for validity). Formally, the set of all policies P is expressed as:

$$P = \{E, R\} \tag{1}$$

where $E = \{E_1, E_2, \ldots, E_n\}$ represents Exigences, and $R = \{R_1, R_2, \ldots, R_m\}$ represents Restrictions.

Each policy P_i is characterized by its weight (W_i) and a set of remarkable states (S_i), ensuring relative importance and evaluative criteria:

$$P_i = \{W_i, S_i\}, \quad \sum_i W_i = 1. \tag{2, 3}$$

Information used in policies can be or Context Elements or Properties. Dynamic **context elements** (C) describe real-time system states [5], such as CPU usage or network latency, while static **properties** (P) represent immutable system characteristics, like CPU cores or device architecture. Together, these elements define the system's operational conditions.

Context elements are structured as:

$$C_i = \{\text{Name}, \text{Value}, \text{Type}\} \tag{4}$$

where **Name** identifies the element, **Value** reflects runtime status, and **Type** specifies its data type. Similarly, properties are defined as:

$$P_j = \{\text{Name}, \text{Value}, \text{Type}\}. \tag{5}$$

In this formalism, all policies are composed by a set of **Remarkable States**, that encapsulate conditions for policy evaluation. A Remarkable state (R_s) is structured as:

$$R_s = \{\text{Operand}, \text{Source}, \text{Operator}, \text{Expected Value}, \text{Tolerance}\}, \tag{6}$$

where **Operand** identifies an attribute, **Source** specifies whether it belongs to **Context** or **Properties**, **Operator** determines the comparison type, **Expected Value** is the target, and **Tolerance** reflects permissible variation.

This original formalism has notable limitations: - **Unified Collection Issue**: Context elements and properties are stored in a single flat collection without logical grouping, making it challenging to distinguish between dynamic and static attributes. - **Sequential Evaluation Overhead**: Policies are evaluated sequentially, requiring all conditions to be checked even if an invalid condition is encountered early, leading to inefficiency. - **No Prioritization**: Policies are treated equally, lacking mechanisms to differentiate between critical and non-critical conditions.

3.2 The Extended Formalism

To address the challenges outlined in Section 3.1, the extended formalism introduces systematic enhancements that improve the organization, prioritization, and evaluation of resource management policies. These enhancements include logical subsets for properties and context elements, explicit policy prioritization, and improved evaluation mechanisms.

In this extension of the original formalism, properties (P) and context elements (C) are now explicitly linked to the logical subsets to which they belong. This structural change ensures better organization, making it easier to evaluate policies and manage attributes.

Properties (P): Properties still represent static, immutable characteristics of the system. However, unlike the original formalism, where properties were stored in a single collection, the extended formalism organizes them into logical subsets ($P_0, P_1, \ldots$) based on their domain (e.g., CPU-related, memory-related). Each property explicitly specifies its subset membership.

Formally:

$$P = \{P_0, P_1, \ldots, P_n\} \tag{7}$$

Each property is defined as:

$$p = (\text{key}, \text{values}, \text{associated\_set}) \tag{8}$$

Example:

$$P_0 = \{(\text{"architecture"}, \text{"ARM"}, \text{"CPU"}), (\text{"physical cores"}, 8, \text{"CPU"})\} \tag{9}$$

Context Elements (C): Context elements still represent dynamic attributes of the system's runtime state. But now, like properties, they are divided into logical subsets ($C_0, C_1, \ldots$) based on their relevance (e.g., CPU-related, network-related). Each context element explicitly references its subset membership.

Formally:

$$C = \{C_0, C_1, \ldots, C_n\} \tag{10}$$

Each context element is defined as:

$$c = (\text{key}, \text{value}, \text{type\_value}, \text{associated\_set}) \tag{11}$$

Example:

$$C_0 = \{(\text{"usage"}, 75, \text{"CPU"}), (\text{"temperature"}, 65, \text{"CPU"})\} \tag{12}$$

By introducing logical subsets, the extended formalism enhances the interpretability and manageability of both static and dynamic information types, directly addressing the ambiguity in the original model.

We also extended the police specification in order to enable flexibility and efficiency in resource management. In order to do that, policies are now divided into

Mandatory ($\mathrm{Pol}_{\mathrm{obg}}$) and **Optional** ($\mathrm{Pol}_{\mathrm{opt}}$) subsets, explicitly prioritizing conditions. In turn, each policy subset remains divided into into requirements and restrictions, like in the original version.

Formally:

$$\mathrm{Pol} = \{\mathrm{Pol}_{\mathrm{obg}}, \mathrm{Pol}_{\mathrm{opt}}\} \tag{13}$$

Finally, the policies now also indicates to which category they belong:

$$p = (\mathrm{Context}, \mathrm{Properties}, W, S_C, S_P) \tag{14}$$

The extended formalism offers several key advantages over the original approach. First, its logical subsets for properties and context elements reduce ambiguity and enable clear policy references. Second, explicitly prioritizing mandatory and optional policies ensures that critical conditions receive precedence. Third, efficiency is greatly enhanced by the fail-fast mechanism, which avoids unnecessary evaluations once a mandatory condition fails. Finally, this extended approach supports scalability in complex and large-scale systems, effectively addressing the original formalism's limitations.

4 Analysis

In this section we compare the original and extended formalisms in terms of both complexity and practical applicability. A collaborative resource-sharing scenario demonstrates how the extended formalism offers clear advantages over the original. A subsequent computational complexity study further highlights improvements made possible by the extended formalism, including parallel evaluation algorithms.

4.1 Application Example: Collaborative Resource Sharing in a Commercial Center

This example explores the application of both formalisms to a scenario involving three companies in a commercial center that collaborate to share computational resources. Each enterprise contributes its resources and defines policies to regulate access. The goal is to ensure that mandatory services are protected while allowing opportunistic services whenever possible, depending on resource availability. The scenario is resumed by Table 4.1

Each company defines policies for managing resources using either the original or extended formalism, as resumed in the Table 4.1

In the **original formalism**, there is no explicit information about whether a remarkable state operates on a context element (C) or a property (P). This ambiguity leads to challenges in correctly interpreting and evaluating policies, particularly for scenarios requiring precise differentiation between static and dynamic attributes. By

Company	Resources	Mandatory Services	Opportunistic Services
Enterprise A	2 notebooks, 3 servers	Fixed, tied to resources	Moderate CPU load, low network latency
Enterprise B	1 server, 10 Raspberry Pis	Low resource tasks, fixed to devices	Lightweight, low memory, moderate network
Enterprise C	5 servers	High computation, tied to specific servers	High computation, requires significant resources

Table 1 Description of enterprises and their resource usage.

Formalism	Policy Example	Original Formalism	Extended Formalism
Enterprise A	Service-Specific Policy	$\{(\text{CPU\_usage}, \leq, 70, 0.1), (\mathit{Network\_latency}, \leq, 50, 0.05)\}$	$\{(\text{CPU\_usage}, C_0, \leq, 70, 0.1), (\mathit{Network\_latency}, \text{C}_1, \leq, 50, 0.05)\}$
	Resource-Specific Policy	$\{(\text{CPU\_cores}, \geq, 4, 0), (\mathit{Available\_memory}, \geq, 8, 0.05)\}$	$\{(\text{CPU\_cores}, P_0, \geq, 4, 0), (\mathit{Available\_memory}, \text{C}_1, \geq, 8, 0.05)\}$
Enterprise B	Service-Specific Policy	$\{(\text{Network\_throughput}, \geq, 50, 0.05)\}$	$\{(\text{Network\_throughput}, C_1, \geq, 50, 0.05)\}$
	Resource-Specific Policy	$\{(\text{Available\_memory}, \geq, 2, 0.05)\}$	$\{(\text{Available\_memory}, C_1, \geq, 2, 0.05)\}$
Enterprise C	Service-Specific Policy	$\{(\text{Available\_CPU\_cores}, \geq, 6, 0.05), (\mathit{Memory\_usage}, \leq, 70, 0.1)\}$	$\{(\text{Available\_CPU\_cores}, C_0, \geq, 6, 0.05), (\mathit{Memory\_usage}, \text{C}_1, \leq, 70, 0.1)\}$
	Resource-Specific Policy	$\{(\text{L3\_cache}, \geq, 16, 0)\}$	$\{(\text{L3\_ache}, P_0, \geq, 16, 0)\}$

Table 2 Comparison of original and extended formalisms for different policies.

contrast, the **extended formalism** explicitly associates each remarkable state with a logical subset, such as C_0 for CPU-related context elements or P_0 for CPU-related properties, ensuring clarity and efficiency in evaluations.

Enterprise C requires policies that involve conditions across multiple servers and subsets. These policies include requirements such as at least six available CPU cores (C_0), memory usage below 70% (C_1), an L3 cache of at least 16 MB (P_0), and CPU usage by mandatory services not exceeding 90% (C_0). Under the original formalism, these policies present significant challenges. All context elements and properties are stored in a unified collection, making it impossible to distinguish between dynamic and static attributes. The lack of logical subsets (C_0, C_1) creates ambiguity and hampers effective policy evaluation.

Furthermore, the original formalism requires all remarkable states to be checked even if a critical failure occurs early in the sequence. For instance, if the CPU cores requirement fails, memory usage and L3 cache conditions are still unnecessarily evaluated, resulting in computational inefficiency. The absence of prioritization mechanisms compounds the problem, as all policies are treated equally, regardless of their criticality.

In contrast, the extended formalism resolves these challenges by introducing logical subsets that explicitly separate attributes into meaningful categories. This structure allows policies to reference subsets directly, simplifying evaluations and eliminating ambiguity. Fail-fast mechanisms terminate evaluations early if a condition fails, reducing computational overhead. Additionally, parallel evaluations enable simultaneous processing of subsets like C_0 (CPU-related) and C_1 (memory-related), significantly improving performance.

4.2 Computational Complexity Analysis

The computational complexity of evaluating policies under the original and extended formalisms is analyzed using the evaluation of a single remarkable state as the fundamental unit of work. Let n represent the total number of remarkable states in the policy set, and p the number of processors available for parallel evaluation.

Under the original formalism, all remarkable states are evaluated sequentially without differentiation between mandatory and optional policies. Even in the best case, all n states are evaluated due to the lack of fail-fast mechanisms. On average, all n states are evaluated, and in the worst case, the process is identical. Thus, the complexity for all cases is $O(n)$.

The extended formalism allows the usage of fail-fast mechanisms, reducing complexity. For sequential evaluation, the best case occurs when a mandatory condition fails immediately, resulting in $O(1)$ complexity. On average, the evaluation involves a fraction of n, yielding $O(n)$ complexity. In the worst case, all states are evaluated sequentially, maintaining $O(n)$ complexity.

Parallel evaluation distributes the workload across p processors. In the best case, a single failure terminates the process, resulting in $O(1)$. On average, the workload per processor is n/p, and the complexity becomes $O(n/p)$. In the worst case, all states are evaluated across processors, maintaining $O(n/p)$ complexity.

Parallel algorithms can theoretically significantly reduce the effective evaluation time, particularly for large-scale systems. Logical subsets enable independent evaluations, maximizing parallelism and further enhancing scalability.

Formalism	**Evaluation Scenario**	**Complexity**
Original Formalism	Best Case	$O(n)$
	Average Case	$O(n)$
	Worst Case	$O(n)$
Extended Formalism (Sequential)	Best Case (fail-fast)	$O(1)$
	Average Case	$O(n)$
	Worst Case	$O(n)$
Extended Formalism (Parallel)	Best Case (fail-fast)	$O(1)$
	Average Case	$O(n/p)$
	Worst Case	$O(n/p)$

Table 3 Computational complexity comparison of the original and extended formalisms under different evaluation scenarios. Here, n is the number of remarkable states and p is the number of available processors.

This comparative analysis demonstrates the extended formalism's superior efficiency, scalability, and adaptability. By addressing the limitations of the original framework, the extended formalism provides a robust solution for managing resources in pervasive systems, making it well-suited for dynamic and large-scale environments.

4.3 Behavior Simulation

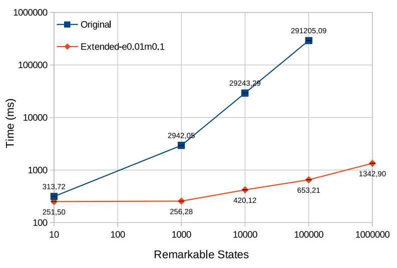
This section presents simulations designed to verify how both formalisms behave under different conditions. We used two C programs—one per formalism—that simulate the execution time of a set of policies defined according to either the original or extended policy-specification approach. Both simulators incorporate random delays to represent practical scenarios, such as observing a context element (via sensor readings), accessing a static property (via a faster table lookup), and marking certain states as invalid based on a predefined invalidation ratio. In the extended formalism, users can additionally specify the ratio of mandatory policies. Both simulators can operate in sequential or parallel mode using OpenMP, and the parallel mode supports dynamic, static, or guided scheduling. All parallel tests ran on a 12th Gen Intel(R) Core(TM) i5-1235U heterogeneous processor featuring two core types, operating at 1.3 GHz and 0.4 GHz, respectively. The simulation codes are freely available in a git repository [1]. Every simulation was executed 30 times.

First, we evaluated how each formalism scales as the total number of remarkable states increases under sequential processing, followed by a test of potential parallel evaluation for both formalisms. In these tests, we varied only the number of remarkable states while keeping all other parameters fixed. The total number of states ranged from 100 to 1,000,000, organized into 10×10, 40×25, 100×100, and 1000×1000 geometries (i.e., Policies $\times$ States). As shown in Figure 1, the extended formalism performs significantly better than the original, largely thanks to its fail-fast mechanism, all while maintaining flexibility in policy specification.

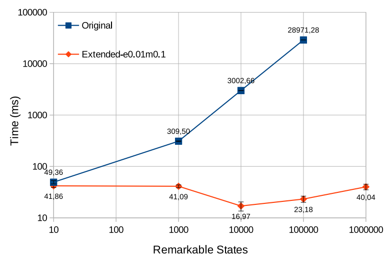
Next, we examined how different invalid-state ratios affect performance in the extended formalism. We tested ratios from 0.01% to 100% in an environment of 1000 policies with 1000 states each. Results presented on Figure 2 shows that even a 10% ratio proved impactful, as the fail-fast mechanism led to substantial time savings by halting early once mandatory states failed. These tests also showed that, for a small number of policies, thread-creation overhead can surpass any parallel speedup benefits, making parallel evaluations less advantageous.

Lastly, we explored the influence of varying mandatory-policy ratios under similar conditions. As illustrated in 3, the outcomes closely matched those in the previous experiment, indicating that the fail-fast mechanism effectively reduces evaluation time even when few states or policies are invalid. Taken together, these results confirm that the extended formalism not only remains efficient in handling both low

[1] https://github.com/dwbeserra/AINA2025FORMALISM

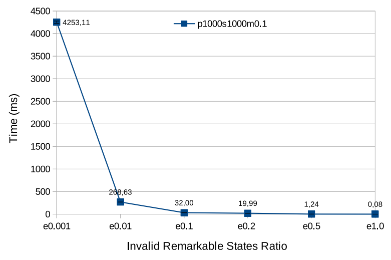


(a) Sequential

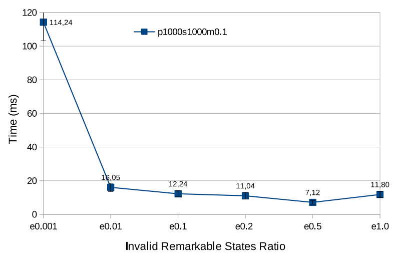


(b) Parallel

Fig. 1 Scalability : original x extended Formalism.



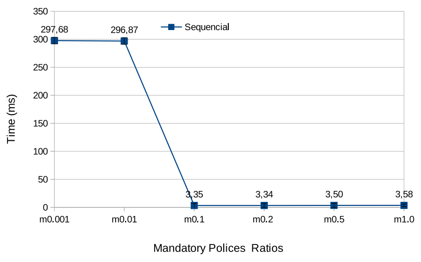
(a) Sequential



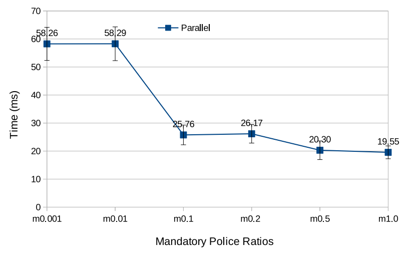
(b) Parallel

Fig. 2 Effects of invalid remarkable states ratio on police evaluation times.

and high invalidation rates but also offers a clear separation between mandatory and opportunistic policies, properly manages context versus property information, and accommodates user-defined requirements and restrictions.



(a) Sequential



(b) Parallel

Fig. 3 Effects of mandatory polices ratio on police evaluation times.

5 Conclusions

This paper introduced an extended formalism for specifying and evaluating resource management policies in pervasive computing environments, addressing significant shortcomings of previous approaches. By distinguishing between mandatory and optional policies, supporting logical subsets of information (static properties versus dynamic context), and incorporating a fail-fast mechanism, the proposed formalism achieves both flexibility in policy specification and efficiency in policy evaluation.

Simulations demonstrated that the extended formalism outperforms its predecessor under various scenarios, including different invalidation rates, mandatory-policy ratios, and workloads. In particular, the fail-fast mechanism ensures that critical violations in mandatory policies are caught early, reducing overhead and accelerating decisions. Meanwhile, the logical separation of static and dynamic attributes clarifies policy definitions and highlights different access costs for context elements versus static properties.

Furthermore, users can incorporate domain-specific organizational knowledge (e.g., cross-company resource usage) and tailor policy categories (mandatory vs. opportunistic) to align with varying quality-of-service requirements. Parallel executions, managed by OpenMP with different scheduling policies, showed that even in large-scale settings, the extended formalism achieves consistent gains, though small workloads sometimes negate the benefits of multithreading.

Overall, the extended formalism provides a robust and adaptable framework for pervasive and dynamic environments, ensuring both performance and clarity in policy management. Future work may involve deeper integration with real sensing infrastructures, adaptive scheduling based on observed data accuracy, and extending the fail-fast mechanism to more nuanced cases, such as partial policy fulfillment or hybrid evaluations.

References

1. Ampririt, P., Higashi, S., Qafzezi, E., Ikeda, M., Matsuo, K., Barolli, L.: A fuzzy-based system for selection of slice service type in 5g/6g wireless networks. In: International Conference on Network-Based Information Systems. pp. 14–24. Springer (2024)
2. Beserra, D.: Vers une Gestion de Ressources Opportuniste et Sensible au Contexte dans les Systèmes d'Information Pervasifs. PhD dissertation, Université Paris 1 Panthéon-Sorbonne (2023)
3. Beserra, D., Pinheiro, M.K., Souveyet, C.: Une gestion de ressources sensible au contexte & opportuniste pour les systèmes d'information pervasifs. In: INFORSID 2021. pp. 123–138 (2021)
4. Breitbach, M., Schäfer, D., Edinger, J., Becker, C.: Context-aware data and task placement in edge computing environments. In: 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom. pp. 1–10. IEEE (2019)
5. Dey, A.K.: Understanding and using context. Personal and ubiquitous computing **5**(1), 4–7 (2001)

6. Islam, M.S.U., Kumar, A., Hu, Y.C.: Context-aware scheduling in fog computing: A survey, taxonomy, challenges and future directions. Journal of Network and Computer Applications **180**, 103008 (2021)
7. Kumar, K.A., Konishetty, V.K., Voruganti, K., Rao, G.P.: Cash: context aware scheduler for hadoop. In: Proceedings of the international conference on advances in computing, communications and informatics. pp. 52–61 (2012)
8. Liu, Y., Qafzezi, E., Ampririt, P., Ohara, S., Barolli, L.: Fbpcqs-fuzzy-based peer coordination quality systems for p2p networks: Implementation and performance evaluation. International Journal of Mobile Computing and Multimedia Communications (IJMCMC) **11**(3), 22–37 (2020)
9. Mahalle, P.N., Dhotre, P.S.: Context-aware pervasive systems and applications, vol. 169. Springer (2020)
10. Nikolopoulos, V., Nikolaidou, M., Voreakou, M., Anagnostopoulos, D.: Context diffusion in fog colonies: Exploring autonomous fog node operation using ectoras. IoT **3**(1), 91–108 (2022)
11. Nikolopoulos, V., Nikolaidou, M., Voreakou, M., Anagnostopoulos, D.: Fog node self-control middleware: Enhancing context awareness towards autonomous decision making in fog colonies. Internet of Things **19**, 100549 (2022)
12. Olaniyan, R., Fadahunsi, O., Maheswaran, M., Zhani, M.F.: Opportunistic edge computing: concepts, opportunities and research challenges. Future Generation Computer Systems **89**, 633–645 (2018)
13. Salehan, A., Deldari, H., Abrishami, S.: Performance evaluation of two new lightweight real-time scheduling mechanisms for ubiquitous and mobile computing environments. Arabian Journal for Science and Engineering **44**(4), 3083–3099 (2019)
14. Weiser, M.: The computer for the 21st century. Scientific American Ubicomp Paper after Sci Am editing (1991)