ChaOS: A Declarative Framework for Operating System Management and Agile Learning

No Author Given

No Institute Given

Abstract. Operating systems are valuable educational tools for understanding core concepts such as process management, memory, file systems, and security. However, user-friendly systems (e.g., Ubuntu, macOS) abstract away complexities that limit technical literacy, while transparent systems (e.g., Arch Linux, Gentoo) expose control at the cost of steep learning curves. This paper introduces ChaOS, a declarative framework that bridges this gap through transparent yet accessible OS management. We derive the TRANSLUCID principles (Technical Responsibility, Automate Native Solutions, Leveled Understanding, Conscious Interaction, Declarative Design) and EXP protocol (Explain, eXemplify, Paginate) to guide system transparency and documentation. The core orchestrator, chaos-cli, ensures system convergence and idempotency. ChaOS features progressive disclosure through "detail levels" and enables in-system documentation, making it a self-documenting OS manager that promotes both technological literacy and robust automation.

Keywords: System management · IaC · Technological Literacy · Education.

1 Introduction

Operating Systems (OS) are foundational pedagogical tools for teaching computer science concepts such as process management, memory allocation, file systems, and security. The OS serves as the environment where theoretical knowledge meets practical application, enabling the teaching of configuring, managing, and understanding a computational environment. However, when exploring Operating Systems, students frequently encounter significant barriers. These challenges typically arise because systems are either designed to be extremely user-friendly by abstracting complexity (the **Black Box** paradigm) or to be highly permissive, granting the user full control (the **White Box** model).

The **Black Box** paradigm targets general users by obfuscating system operations behind polished interfaces. While enhancing accessibility, this approach hides the causal link between user action and system response [10]. A similar phenomenon is observed in Machine Learning models [3], where "Black Box" opacity prevents users from forming the mental models necessary for system comprehension [6].

Conversely, the **White Box** paradigm targets users seeking customization, directly exposing internal mechanisms to maximize control. However, this raw exposure often exceeds the user's processing capacity, leading to cognitive overload [11] and impeding learning. In particular, within Linux-based distributions, students frequently encounter barriers due to the fragmented and often inaccessible nature of existing documentation. Traditional "man pages," while exhaustive in scope, adhere rigidly to a reference-oriented format [7] characterized by dense technical nomenclature that offers minimal pedagogical utility for newcomers. Web-based documentation, despite its ostensibly greater accessibility, presents distinct limitations: the prevalence of inconsistent terminology across disparate sources necessitates that users synthesize information from multiple, frequently unreliable platforms (community forums and AI-assisted tools) merely to acquire foundational knowledge [1].

To bridge this gap, we propose **Ch-aOS** (an acronym for "**Ch**ange **an** **OS**" and a play on "Chaos"), a framework designed to centralize OS management and didactic documentation. Drawing upon the *Brasil Participativo*[1] IaC infrastructure, the declarative paradigms of *NixOS*[2], and the concept of progressive disclosure, Ch-aOS mitigates the trade-off between usability and control. By integrating system automation with an embedded educational layer, the framework effectively renders system configuration as documentation, implementing the authorial principles of **TRANSLUCID** (Technical Responsibility, Automate Native Solutions, Leveled Understanding, Conscious Interaction, Declarative design) and **EXP** (Explain, eXemplify, Paginate).

2 Study Design

This study adopts the Design Science Research (DSR), a methodology that centers on the creation and evaluation of innovative IT artifacts [4]. We developed the Ch-aOS artifact using an Agile Iterative and Incremental model. This approach facilitated a continuous feedback loop, driving the evolution of the project from a monolithic automation script into a cohesive, modular architecture.

To validate the framework's viability and adherence to the proposed principles, this work was conducted within a standard Arch Linux environment, chosen for its high malleability. This allowed us to assess the system's convergence and didactic features in a real-world scenario, resulting in a robust core called *Ch-aronte*. In subsequent iterations to study the plugin system's viability, we developed two functional models: *chaos-dots*, a dotfiles manager inspired by *Chezmoi* [3], and *chaos-secrets*, a secret-first template manager inspired by *sops-nix* [4].

[1] https://docs-lappis-unb-decidimbr-infra-5774a88ab47379a218e677e5c9a22be.gitlab.io/
[2] https://nixos.org
[3] https://www.chezmoi.io/
[4] https://github.com/Mic92/sops-nix

3 Results

The Architecture: Ch-aOS employs a modular monolith architecture (See Figure 1). While microservices offer Separation of Concerns (SoC), recent literature suggests they introduce liabilities such as service over-extension and functional overlap [2]. To mitigate these issues, Ch-aOS is composed of distinct OS cores, plugins, and a unified monolithic core. The central component, chaos-cli, is a Python-based orchestrator that serves as the primary interface between the user and the system state. Task execution — encompassing *roles*, role *aliases*, and didactic *explanations* — is delegated to an integrated entry_point subsystem. The framework operates on a "Declarative Through Convergence" paradigm. Unlike imperative approaches that may attempt indiscriminate application of configurations, Ch-aOS verifies the existing system state and applies only the necessary deltas to achieve the target configuration, thereby ensuring idempotency.

To better visualize the architecture, a level 2 C4 diagram was made:

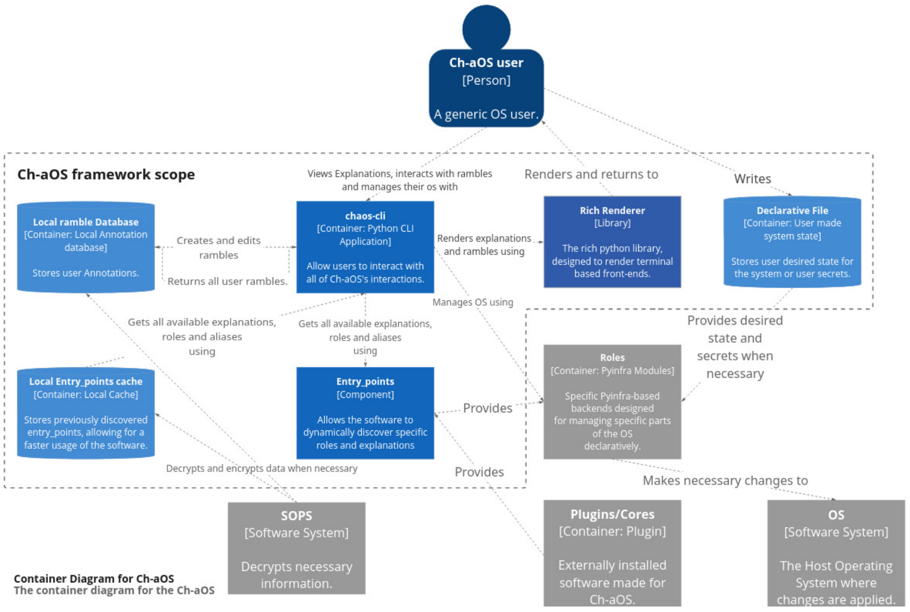
Ch-aOS user
[Person]
A generic OS user.
Ch-aOS framework scope
Views Explanations, interacts with rambles and manages their os with
Renders and returns to
Writes
Local ramble Database
[Container: Local Annotation database]
Stores user Annotations.
Creates and edits rambles
Returns all user rambles.
chaos-cli
[Container: Python CLI Application]
Allow users to interact with all of Ch-aOS's interactions.
Renders explanations and rambles using
Rich Renderer
[Library]
The rich python library, designed to render terminal based front-ends.
Declarative File
[Container: User made system state]
Stores user desired state for the system or user secrets.
Manages OS using
Gets all available explanations, roles and aliases using
Gets all available explanations, roles and aliases using
Provides desired state and secrets when necessary
Local Entry_points cache
[Container: Local Cache]
Stores previously discovered entry_points, allowing for a faster usage of the software.
Entry_points
[Component]
Allows the software to dynamically discover specific roles and explanations
Provides
Roles
[Container: Pyinfra Modules]
Specific Pyinfra-based backends designed for managing specific parts of the OS declaratively.
Decrypts and encrypts data when necessary
Provides
Makes necessary changes to
SOPS
[Software System]
Decrypts necessary information.
Plugins/Cores
[Container: Plugin]
Externally installed software made for Ch-aOS.
OS
[Software System]
The Host Operating System where changes are applied.
Container Diagram for Ch-aOS
The container diagram for the Ch-aOS

Fig. 1. Ch-aOS architecture diagram

The Stack: We used an agentless Python-based library, Pyinfra [5], and the omegaconf [6] library to load the declarative file, adding another layer of robust-

[5] https://pyinfra.com/
[6] https://omegaconf.readthedocs.io/

ness and idempotency on top of the existing ones. We also used SOPs [7] as the main encryption engine, and Rich with Argparse as its “ui” libraries.

The Philosophy: We propose two novel philosophies to guide development and documentation: **TRANSLUCID** (**T**echnical **R**esponsibility, **A**utomate **N**ative **S**olutions, **L**eveled **U**nderstanding, **C**onscious **I**nteraction, **D**eclarative design) Aligns Ch-aOS with the Unix philosophy of composition [5] and text-based software [8]. Instead of reinventing tools, it focuses on orchestrating native solutions through a declarative design, enforcing user responsibility and maintains a clean implementation.

EXP (**E**xplain, e**X**emplify, **P**aginate) mandates that concepts need to be Explain-ed to help with mental models [6], eXemplifi-ed (both manual and Ch-aOS-abstracted examples) to implement the Worked Example Effect [11], and Paginat-ed to ensure progressive disclosure [9]. Crucially, EXP mandates a *bidirectional flow*: users can annotate concepts within the system, creating a feedback loop between automation and learning. This creates novel documentation ideals, falling between "didactic" and "direct" [7].

4 Discussion and Conclusion

We introduced Ch-aOS to resolve the disparity between “Black Box” automation and “White Box” transparency [10]. By implementing a **Glass Box** approach through TRANSLUCID and EXP, our framework restores the causal link lost in technological opacity while mitigating cognitive load through progressive disclosure. Decoupling execution from explanation allows users to interact at their own pace, fostering technological literacy alongside automation.

The development of the **chaos-cli** orchestrator validated that state-of-the-art automation tools can be repurposed for educational intent while preserving idempotency. The migration to a modular architecture based on Pyinfra improved code clarity and debugging compared to monolithic scripts or YAML-first solutions [2]. Furthermore, the use of Python facilitated “Conscious Interaction,” enabling the verification of user intent before system convergence. However, the Glass Box approach requires active maintenance. We resolved ecosystem drift (e.g., Pyinfra updates) by packaging the core via **PEX** and addressed pedagogical gaps with a built-in note-taking CLI, chaos ramble. To reduce the initial setup friction created by the declarative layer, we also developed an **init** system.

Our future work focuses on adapting to Python ecosystem changes and developing a NixOS core named **Ch-imera**. Additionally, we plan a user study to measure the impact of the EXP approach on the learning curve, aiming to validate Cognitive Load Theory in a real-world environment [11].

References

1. Aghajani, E., Nagy, C., Vega-Márquez, O.L., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M.: Software documentation issues unveiled. In: 2019

[7] https://getsops.io/

IEEE/ACM 41st International Conference on Software Engineering (ICSE). pp. 1199–1210. IEEE (2019)

2. Daniel, G., Mota, E., Wang, Y., Guerra, E.: Architecture refactoring towards service reusability in the context of microservices. In: Proceedings of the 2025 Conference on Software Architecture (2025)
3. Garouani, M., Mothe, J., Barhrhouj, S., Aligon, J.: Investigating the duality of interpretability and explainability in machine learning. arXiv preprint arXiv:2403.XXXXX (2024)
4. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS quarterly pp. 75–105 (2004)
5. McIlroy, M.D., Pinson, E.N., Tague, B.A.: Unix time-sharing system: The unix shell. The Bell System Technical Journal **57**(6), 1991–2019 (1978)
6. Norman, D.: The Design of Everyday Things: Revised and Expanded Edition. Basic Books (2013)
7. Procida, D.: Diátaxis: A systematic approach to technical documentation authoring. https://diataxis.fr (2021), accessed: 2025-12-08
8. Raymond, E.S.: The Art of Unix Programming. Addison-Wesley Professional (2003)
9. Springer, A., Whittaker, S.: Progressive disclosure: empirically motivated approaches to designing effective transparency. In: Proceedings of the 24th International Conference on Intelligent User Interfaces. pp. 107–118 (2019)
10. Surden, H., Williams, M.A.: Technological opacity, predictability, and self-driving cars. Cardozo L. Rev. **38**, 121 (2016)
11. Sweller, J.: Cognitive load during problem solving: Effects on learning. Cognitive science **12**(2), 257–285 (1988)