iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** III **Month of publication:** March 2026

DOI: https://doi.org/10.22214/ijraset.2026.77757

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

VoXlinux: A Voice-Driven Linux Operating System for Secure and Hands-Free Operation

Kamatchi T P[1], Abhijay S[2], Lakshith S[3], Mohammad Ishaan M[4], Thaufiq N H[5]
[1]*Head of the Department,* [2, 3, 4, 5]*Students, Department of Computer Engineering, PSG Polytechnic College*

Abstract: ***VoXLinux is a Linux-based system enhancement built on Arch Linux that combines autonomous background self-healing, user-controlled intent execution and a safe conversational assistance module to improve system reliability and usability. The system continuously monitors system services, running processes, boot integrity, and package states through a dedicated self- healing daemon that which is designed and implemented using a deterministic state-driven architecture. This daemon actively detects failed, locked, inconsistent, or stalled components such as broken systemd units, pacman database locks, and initramfs inconsistencies and restores them using predefined, non-destructive recovery policies without requiring user intervention. To ensure operational safety, the system is structured with the recovery engine around confidence scoring and gated healing stages, preventing unsafe or ambiguous automatic actions. High-level administrative control is provided through an intent interface that can be explicitly enabled or disabled using secure keybindings and accessed via voice commands, ensuring that critical system modifications occur only with clear user awareness and authorization. The intent engine follows a deterministic executionpipeline, translating user instructions into verified system operations while maintaining explainability and auditability. In addition toita lightweightconversationalassistantthat operates andgeneratingconcise systemsummaries andguidance responseswithout executing commands or influencing the healing subsystem. By clearly separating autonomous recovery, intent-driven administrative control, and non-executing conversational assistance, VoXLinux reduces manual maintenance effort, minimizes downtime, prevents unsafe automation, and introduces a structured, policy-governed approach to Linux system management suitable for real-world and research-oriented deployment environments.***

Keywords: Linux Operating System, Self-Healing Systems, Intent-Based Control, System Recovery, Arch Linux, Automation, Explainable Systems

I. INTRODUCTION

Linux is widely adopted as a server, desktop, and embedded operating system due to its performance efficiency, security model, modular architecture, and open-sourceecosystem. Its flexibility enables deployment across enterpriseservers, researchsystems, cloud infrastructure, and consumer devices. However, maintaining system stability and reliability requires considerable technical expertise. Administrative tasks such as managing system services, handling failed package upgrades, resolving dependency conflicts, and recoveringfromboot-time errors oftendemandmanualinterventionanddeepfamiliaritywithlow-leveltools.System-levelissues such as malfunctioning systemd services, interrupted or corrupted package database states, broken initramfs configurations, or unsafe filesystem conditions can render a system unstable or even unbootable. In traditional Linux environments, diagnosing and resolving such failures is reactive and command-driven, increasing downtime and the risk of improper recovery actions. VoXLinux addresses these limitations by introducing a structured self-healing abstraction layer built on top of a standard Arch Linux system. Instead of relying solely on manual troubleshooting, the system continuously monitors critical runtime components, evaluates service health, verifies package consistency, and assesses filesystem safety conditions. When failures or inconsistencies are detected, deterministic and non-destructive recovery policies are applied automatically through a controlled state-driven healing mechanism. This reduces administrative burden and improves operational reliability.Beyond autonomous recovery, VoXLinux introduces an intent-basedsystem control modelthat replaces low-levelcommand execution with high-level user intentspecification. Rather thanrequiring users to remember and execute complex command sequences, the framework allows them to express desired outcomes, which are then translated into verified and explainable system actions. This approach improves usability while maintaining strict safety constraintsand execution transparency. By combining autonomous background healing with optional intent-driven administrative control, VoXLinux aims to enhance system resilience, reduce downtime, and provide a safer and more user-centric Linux management paradigm suitable for practical deployment and research environments.

©IJRASET: All Rights are Reserved

II. LITERATURE SURVEY

This idea of self-managing systems originates from autonomic computing, introduced by Kephart and Chess (2003), which defined systems capable of self-configuration, self-healing, self-optimization, and self-protection using the Monitor–Analyze–Plan–Execute(MAPE)feedbackloop.Thisframeworkestablishedthefoundationforautonomoussystembehaviour.Subsequent work expanded this vision. Huebscher and McCann (2008) emphasized policy-based adaptation and context-aware decision making, highlighting the separation of high-level objectives from low-level execution. Ganek and Corbi (2003) focused on autonomic infrastructure capable of maintaining stability with minimal human intervention, while van Hoorn et al. (2012) identified runtimemonitoring,model-drivendecisions,andcontrolledexecutionascoreelementsofself-adaptivesoftware.

Automated recovery has been studied through failure-oblivious computing (Tang et al., 2013) and self-healing mechanismsindistributedsystems(Silvaetal.,2012).Whiletheseapproachesdemonstrateresiliencethroughmonitoringandreconfiguration ,they lackdeterministic,intent-drivencontrol,particularlyattheoperatingsystemlevel. Intent-based computing addresses this gap by expressing user goals as declarative specifications. Deutsch (2020) formalized intent as desired outcomes rather than procedures, a concept widely adopted in networking (Feamster et al., 2018). However, intent-driven paradigmshavenotyetbeenfullyrealizedingeneral-purposeoperatingsystems.

In Linux systems, reliability and recovery largely rely on administrative tools such as systemd for service supervision and pacman for package consistency. Recovery from system failures typically requires manual intervention, and although snapshot-based rollbackandtransactionalupdatesimproverecoveryspeed,theyremainreactiveandcommand-drivenratherthanautonomous.Finally, software architecture principles—modularity, separation of concerns, and controlled component interaction—outlined by Parnas (1972),Tayloretal. (2009),andBass etal.(2012)support the designoflayeredcontrolarchitectures thatclearlyseparate observation, decision-making, and execution.

III. PROPOSEDSYSTEM

A. Platform Foundation

VoxLinux is built on Arch Linux, leveraging its minimal base and rolling-release control model to ensure deterministic package and service management. The graphical environment runs on Hyprland within the Cealestia configuration, providing amodern Wayland-native compositor suitable for low-overhead desktop operation. All graphical components of the system —including the intent interface and chatbot window — are implemented using GTK, ensuring consistent rendering, native Wayland compatibility,andmodularUIisolation.Theplatformremainslightweight,composable,andexplicitlycontrolledateverylayer.

B. Proposed System:IntentEngine

1) *Overview:* The Intent Engine is a user-space service responsible for translating explicit user commands into structured, deterministic system actions. It follows a controlled wake–sleep lifecycle to ensure minimal idle resource consumption and complete user-driven activation. The engine does not possess privileged healing authority and operates strictly within predefined execution boundaries.

*C. Wake DetectionLayer:*The engine remains inanidle monitoringstateuntila predefinedwake keyis detectedthrougha keybinding listener integrated with the compositor environment. This mechanism ensures that activation occurs only upon explicit user intent. No continuous voice streaming or background inference is performed during idle operation, preserving system efficiency and preventing unintended processing.

2) *Voice Input Processing:* Upon activation, the engine captures audio input and processes it using Whisper for speech-to-text transcription. The transcribed output is converted into structured textual data suitable for downstream interpretation.Systemfeedbackisdeliveredusing Piper,enablingimmediateconfirmationofrecognizedcommandsandexecutedactions.BothWhisperandPiperOperatestrictly as input–outputtransformers and have no authority over execution logic.

3) *Intent Parsing and Validation:* The transcribed input is forwarded to the intent parsing module, which converts natural language into a machine-processable intent representation. This structured format categorizes the request into predefined intent classes aligned with supported system operations.The validation stage ensures syntactic correctness, contextual consistency, and safety compliance. Permission boundaries are verified before execution eligibility is granted. Only validated and recognized intents are allowedto proceed to routing.

4) *Intent Routing and Execution:* The routing component maps the validated intent to a predefined execution handler. Each handler correspondstocontrolleduser-leveloperationssuchasapplicationlaunching,filehandling ,orboundedautomation routines.

Execution is deterministic and policy-restricted. The engine does not modify system policies, perform recovery operations, or access privileged repairmechanisms.Allactionsareloggedandremainauditable.Followingexecution,feedbackisprovidedthroughthe GTKinterface and text-to-speech confirmation, maintaining clarity and transparency for the user.

5) *Sleep Transition:* After completing an interaction cycle, the engine evaluates whether continued engagement is required. If no additional commands are detected, it transitions back to its idle monitoring state.This structured transition ensures efficient resource utilization, predictable behavior, andexplicit user control.The wake–sleeplifecycle establishes a responsive yet controlled interaction framework that aligns system behavior directly with user intent while maintaining strict operational boundaries.

D. ProposedSystem:Self-HealingLayerArchitecture

TheVoXLinuxself-healinglayerisimplementedasaprivilegedbackgrounddaemon(voxlinuxd)thatprovidesdeterministic, policy-controlled system recovery. The architecture is modular and structured around the coordinated interaction of the observer, probe, system_state, health, healing_level, verifier, pacman,predictive, andexplain modules.The design enforces boundedautonomy, strict safety gating, and reproducible recovery decisions.

1) *Observer and System State Modelling:* The observer module is responsible for structured system inspection. It invokes the probe subsystem to collect telemetry from systemd unit states, pacman database conditions, boot logs via journald, filesystem mount integrity, and active process context. The collected data is consolidated into the system_state abstraction, producing an immutablestate snapshot for each evaluation cycle. The resulting ObserverReport encapsulates failed units, pacman lock status, boot anomalies, filesystem safety indicators, and classified failure metadata. The state representation remains constant during a decision cycle toensure deterministic downstream evaluation. No dynamic state mutation occurs once evaluation begins.
2) *Failure Classification via health Module:* The health module performs rule-based failure classification using the structured system_state. Failures are categorized into predefined classes, including service instability, pacman inconsistency, boot anomaly, and unsafe filesystem condition. Classification relies on invariant checks, threshold validation, and cross-signal consistency analysis. No probabilistic inference or machine learning model is used in determining failure class.Each classified failure maps to a bounded recovery template that defines permissible corrective actions. This ensures that every recovery path is predefined and verifiable.
3) *Confidence ScoringMechanism:*Aboundedhealingconfidence score iscomputed as:

$$C=f(F,S,R,)$$

WhereFdenotestheclassifiedfailuretype,Srepresentssignalstrengthacrossindependentprobes,Rreflectsreproducibilityacross observation cycles,T denotes state consistency constraints within system_state.Healing is permitted only if:

$$C\geq Cthreshold$$

and no violations are detected by the healing level policy checks. The confidence value is explicitly capped to prevent uncontrolledescalation of autonomy. The predictive and explain modules do not influence the numerical value of C.

4) *Deterministic Repair Construction and Execution:* When recovery conditions are satisfied, the verifier module constructs a deterministic execution plan derived from static recovery templates. The plan includesexplicitpreconditions, ordered execution steps, and post-execution verification routines. For pacman-related failures, the pacman module performs additional validation, including lock-fileinspection,processactivityconfirmation,anddatabaseintegritychecksbeforepermittingcorrectiveaction.Executionisgated bythehealing_levelcontroller,whichvalidatesconfidencethresholds,policyconstraints,bootcontextsafety,anddenylist restrictions. Onlyifallcheckssucceeddoesthesystempermitcontrolledactuation.Eachactionislogged,andpost-actionvalidationismandatory. If verification fails, further attempts are blocked and escalation logic is triggered.High-risk operations require explicit confirmation through the controlled execution interface. A single-attempt-per-session policy prevents instability amplification.
5) *Boot Context and Runtime Control:* Boot context is evaluated during each decision cycle to determine whether the system is in a stable multi-user state, emergency shell, or early initialization phase. Autonomous actuation is restricted to verified safe runtime contexts. Recovery attempts in unstable boot phases are disallowed to prevent cascading failures.For isolated service-level failures with high confidence, a bounded restart attempt is permitted, followed by immediate health re-evaluation through the observer and health modules. Repeated automatic retries are disallowed.

6) *Security and Integrity Guarantees:*The architecture enforces deterministic decision paths, strict privilege separation, bounded AI involvement, and comprehensive audit logging. All state transitions require post-action validation, and rollback capability must be confirmed before high-impact operations are executed. Recovery strategies prioritize non-destructive actions and enforce policy compliance at every execution boundary.

E. ProposedSystem:ConversationalAssistance Module

The system integrates a lightweight conversational chatbot designed to provide immediate, concise support for userqueries within the VoxLinux environment. This module functions as a learning-oriented assistance layer that helps users understand Linux operations, development concepts, and general technical workflows through short, context-aware responses. It is implemented as a user-space component with a GTK-based pop-up interface, allowing on-demand interaction without interrupting the primary desktop workflow. The chatbot performs only basic conversational tasks and informational guidance, operating within clearly defined functional limits and without any authority over system control, intent execution, or self-healing processes.

User input is forwarded to a remote language model through the Groq API using a secure, environment-based authentication mechanism. The returned responses are formatted locally and presented in a minimal, distraction-free chat window designed for quick reference and continuous learning. The module maintains session-level context to support natural interaction flow but avoids persistent storage of conversational data. Its purpose is strictly educational and assistive, enabling users to obtain quick explanations, command references, and conceptual clarity in real time. By remaining modular, non-privileged, and operationally isolated, the chatbot enhances usability and accessibility while preserving the deterministic and policy-controlled behavior of the core system.

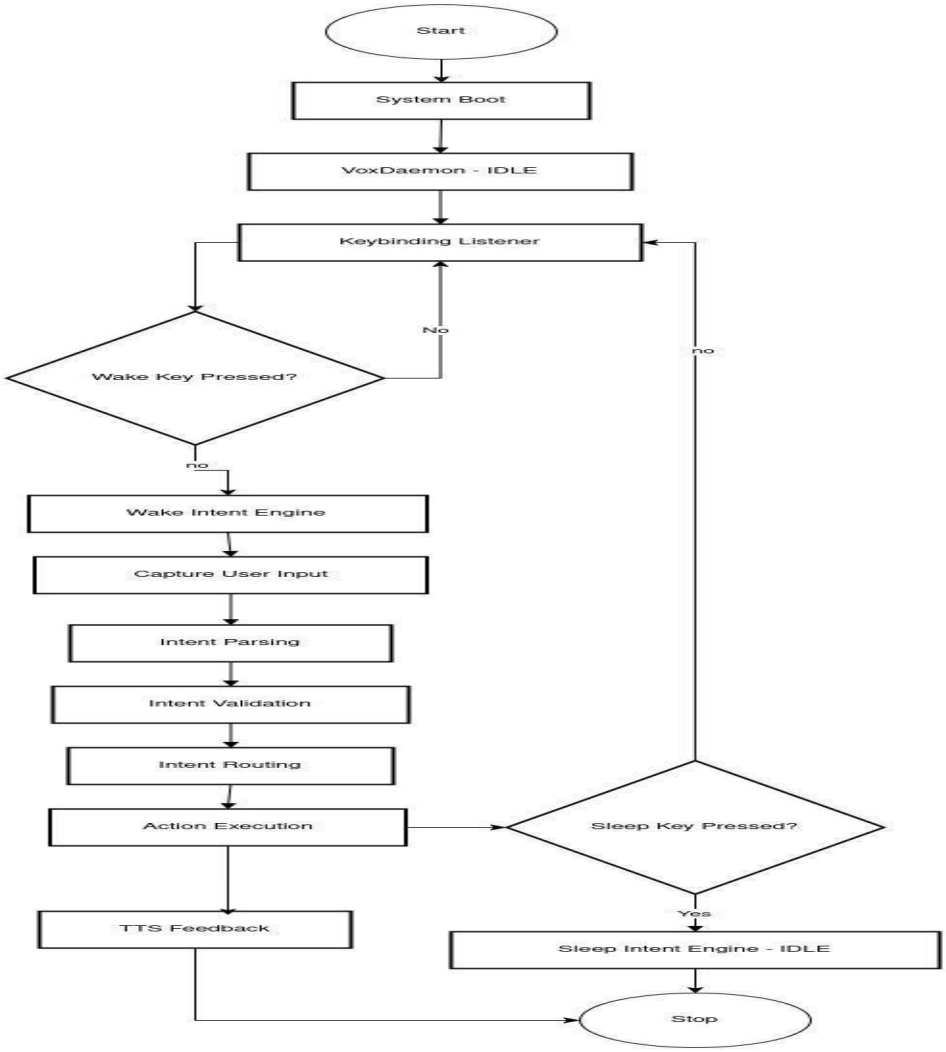


Fig.1Intent Engine

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

IV. RESULT ANALYSIS

The performance of the VoxLinux intent–driven engine was analysed under continuous daemon execution to evaluate activation latency, intent recognition accuracy, execution control, runtime configurability, and system resource usage. The VoxDaemon remained in a low-overhead passive state and responded immediately to wake-key events, confirming that the event- based keybinding mechanism enables real-time interaction without affecting foreground user activities. Captured input was consistently processed through the intent management pipeline, where tokenization, classification, and confidence scoring produced reliable intent identification for predefined command sets. The confidence-regulated routing layer effectively blocked low-certainty andinvalidintents,ensuringthat onlyvalidatedactions reachedthe executionstage.Dynamicconfiguration reloadingapplied updated intent mappings and key definitions during active operation without requiring a service restart, demonstrating live behavioural adaptability. All system operations were performed through the system adapter abstraction, which isolatedexecution failures from the semantic processing flow and maintained stable daemon behaviour across repeated interaction cycles. The text-to-speech feedback mechanismgeneratedimmediateauditoryconfirmationfor successful,rejected,andunmappedintents,completinga closedinteraction loop, while prolonged runtime observation showed consistently low CPU and memory consumption suitable for real-time desktop deployment.

TABLE1
USERINTERVENTIONANALYSIS

Scenario	Detection	Recovery	User Intervention
Servicefailure	High	Automatic Restart	None
Processstall	High	Automatic Relaunch	None
Safe package lock	High	Automatic	None
Administrative operation	Intent-based	User triggered	Required
User query	Not applicable	Concise response	None

TABLE2
COMPARATIVEOBSERVATION

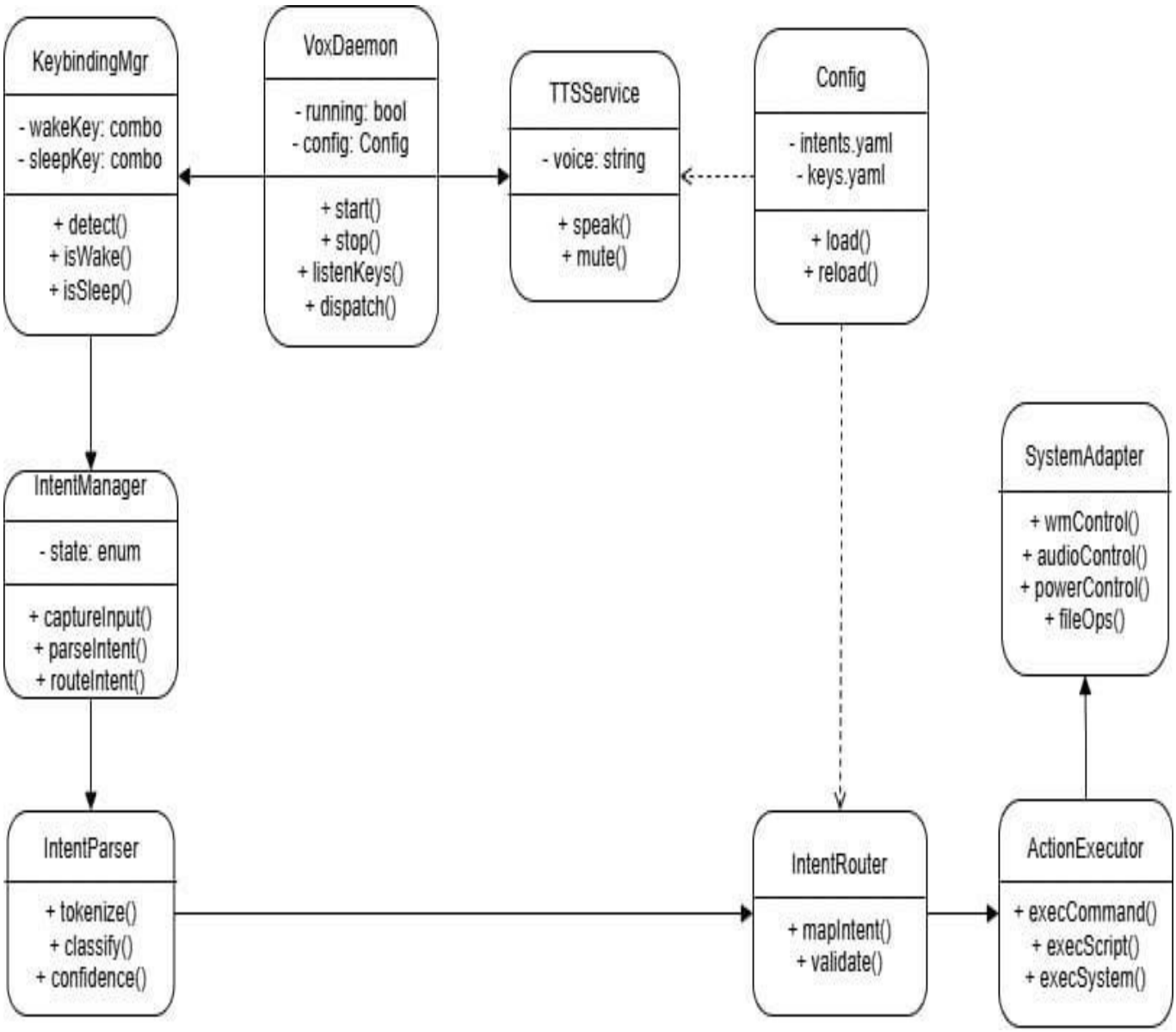
Feature	TraditionalMethod	VoXLinux
ServiceRecovery	Manual command execution	Automaticbackground healing
Processmanagement	User monitoring required	Autonomousdetection and relaunch
Administrativecontrol	Always command	User-gated intent execution
Conversationalinterface	Direct command execution	Summarization only (safe)
Systeminterruption	High	Minimal thank

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

A. Discussion

The results indicate that the proposed architecture replaces conventional command-driven interaction with a controlled semantic execution model in which interpretation, validation, and system control are structurally separated. Confidence-based gating functions as the primary safety layer by allowing execution only when semantic certainty satisfies predefined thresholds, thereby preventing unintended operations while preserving user authority through explicit activation. The persistent daemon design eliminates repeated process initialization andenablesinstantaneous responsiveness, making the framework practical for continuoususe. Runtime configuration reloadability converts the system from a static control mechanism into a dynamically extensible platform capable of incorporating new intents and workflows without operational disruption. The system adapter introduces a secure execution boundary that ensures predictable and fault-isolated behaviour even when invalid input is received. In addition, the integrated feedback channel transforms system control into a bidirectional interaction process, improving usability and transparency. Although the current implementation operates within a predefined intent scope, the modular separation of parsing, routing, and execution layers provides a clear path for future integration of adaptive inference models, establishing VoxLinux as a stable, scalable, and evolution-ready semantic interface for Linux environments.



©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

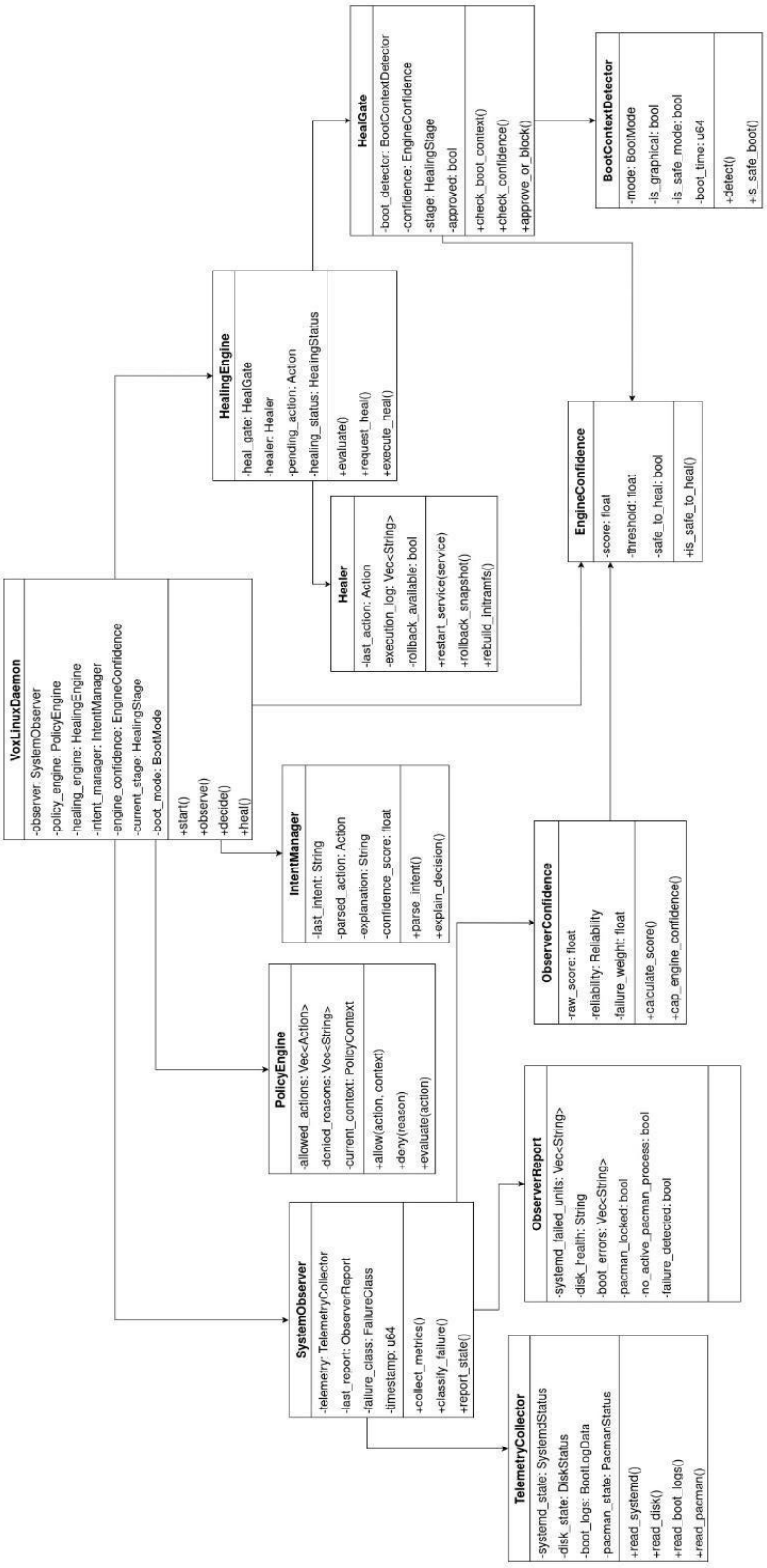


Fig.3Self-Healing

V. CONCLUSION AND FUTURE WORKS

A. Conclusion

VoXLinux enhances the conventional Linux operating environment through a structured layered architecture that unifies autonomous recovery, intent-driven administrative control, and a secure conversational interface into a single continuous runtime. In contrast to traditional workflows that depend on manual command execution for diagnosing and restoring failed services or stalled processes, the system introduces a persistent self-healing daemon that continuously observes runtime behaviour and performs safe, non-destructive corrective actions in the background. Failed systemd units are automatically restarted, unresponsive processes are cleanly terminated and relaunched, and package manager lock conditions are resolved through controlled recovery procedures that preserve database integrity. These operations are performed without interrupting active user tasks, thereby reducing downtime while maintaining a stable working environment. At the same time, the intent-based execution model ensures that sensitive administrative operations are never triggered autonomously; they require explicit user activation through predefined keybindings or validated voice input, preserving human authority over critical system changesand eliminating the risk associated with uncontrolled automation. The conversational assistant complements this architecture by delivering concise, human-readable system summaries without invoking privileged execution paths, which prevents accidental or unauthorized command execution. The strict functional separation between the healing engine, intent router, execution layer, and chatbot module establishes clear security boundaries, enabling predictable behaviourandfaultisolation.Experimentalobservationshows thatthis approachsignificantlyreduces manualintervention,maintains consistently low resource consumption during prolonged daemon operation, improves service availability, and provides a more accessibleanduser-centricmethodforsystemmanagement, demonstratingtha tVoXLinuxoperatesasa practicalanddeployableself- stabilizing Linux environment rather than a purely conceptual framework.

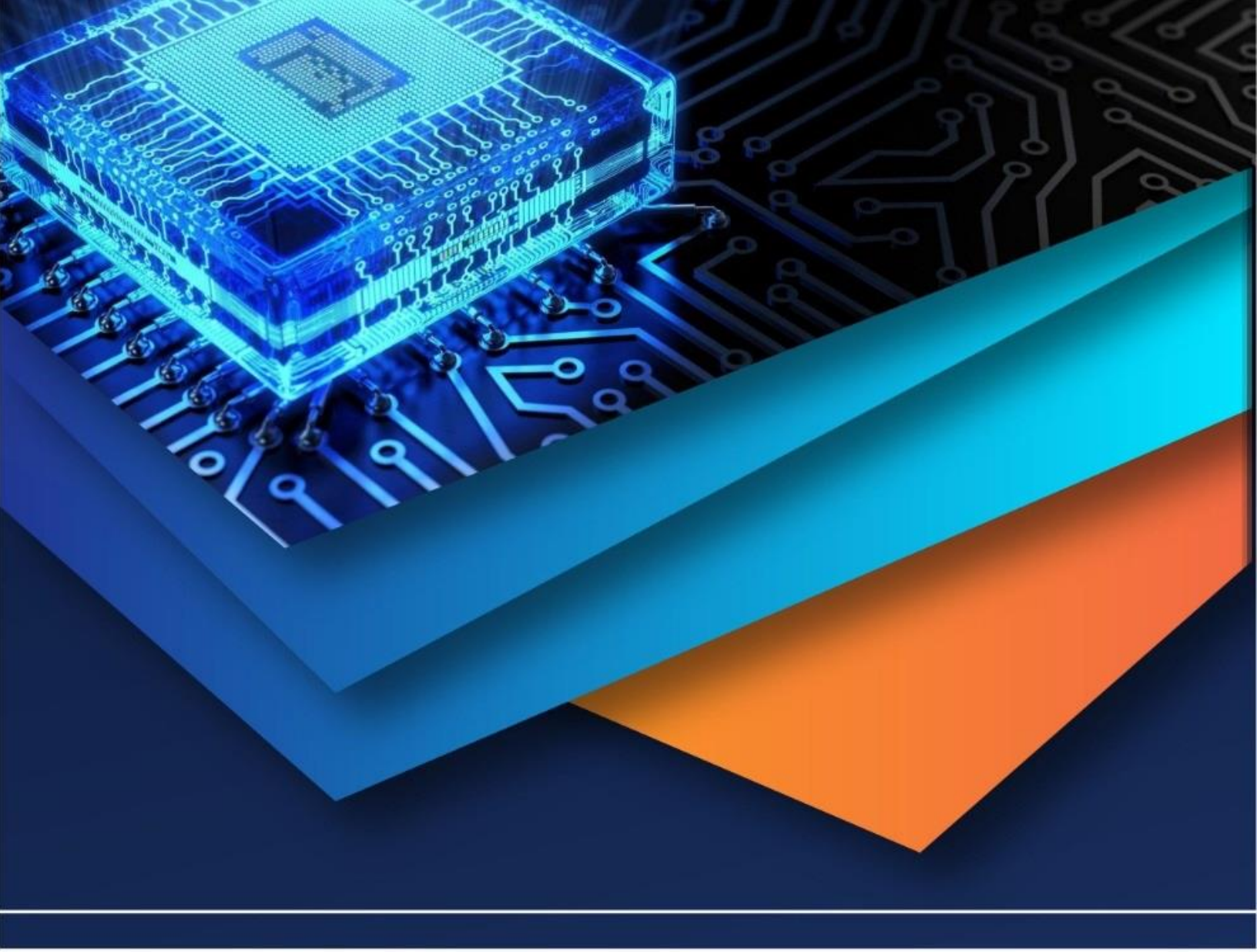
B. FutureWork

The future evolution of VoxLinux is directed toward transforming the current intent-driven architecture into a context-aware and predictive system management platform capable of preventive intervention rather than reactive recovery. This includes the integration of automated filesystem integrity analysis and intelligent network service stabilization to detect latent inconsistencies and restore operational reliability before user impact occurs. Log-based predictive failure analysis will extend the monitoring layer by leveraginghistoricalsystemtelemetrytoidentifybehaviouralpatternsthatprecederuntimefaults,enablingearlyandpolicy-controlled remediation. A configurable policy framework will allow users to define recovery strategies, trust thresholds, and staged response models that adapt to system load, operational priority, and execution risk, thereby converting the healing mechanism into a user-governed decision system. Portability will be improved through distribution-agnostic abstraction of package management and service control interfaces, allowing the platform to operate consistently across multiple Linux ecosystems. The introduction of a real-time visual health dashboard will make internal system states observable by presenting service conditions, intent confidence levels, and recovery decisions in an interpretable form, enhancing transparency and administrative awareness. The voice interaction layer will be further optimized for fully offline processing and multilingual support to ensure privacy, inclusivity, and reliable operation inrestricted environments. In addition, high-risk administrative tasks will be safeguarded through secure rollback mechanisms and comprehensive audit trails that provide verifiable execution history and accountability. Together, these enhancements reposition VoxLinux from a reactive self-healing environment to an adaptive, policy-driven, and explainable control layer for Linux systems, capable of learning from operational history and responding intelligently to evolving runtime conditions.

REFERENCES

[1] J.O. Kephartand D.M. Chess,“Thevisionofautonomiccomputing,”Computer,vol. 36, no. 1,pp. 41–50, Jan. 2003. M.C.HuebscherandJ.A.McCann,“Asurveyofautonomiccomputing—Degrees,models,andapplications,”ACMComputingSurveys,vol. 40,no. 3, pp.1–28, Aug. 2008.
[2] A.G.GanekandT.A.Corbi,“Thedawningoftheautonomiccomputingera,”IBMSystemsJournal,vol.42,no.1,pp.5–18, 2003.
[3] L.P.Deutsch,“Aninteractive intent-basedcomputingmodel,” Communications ofthe ACM,vol.63,no.2,pp.46–54,Feb. 2020.
[4] ArchLinuxDevelopers,“ArchLinuxdocumentation,”ArchLinux,2024.[Online].Available: https://archlinux.org
[5] L.Poettering,“systemdforadministrators,”LinuxJournal,2013.
[6] S.Kroah-Hartmanand A.Corbet,Linux KernelDevelopment,3rd ed.Indianapolis,IN, USA:Addison-Wesley, 2010.
[7] L.Bass, P. Clements, andR. Kazman, SoftwareArchitecture in Practice, 3rd ed.Boston, MA,USA:Addison-Wesley, 2012.
[8] M.Fowler,Patterns ofEnterprise ApplicationArchitecture.Boston,MA,USA:Addison-Wesley,2002.
[9] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, Software Architecture:Foundations, Theory, and Practice. Hoboken, NJ, USA: Wiley, 2009.
[10] E.Gamma,R.Helm,R. Johnson, andJ. Vlissides,DesignPatterns:Elementsof ReusableObject-OrientedSoftware. Boston, MA, USA: Addison-Wesley, 1994.
[11] R.Love,LinuxSystemProgramming.Sebastopol,CA,USA:O’ReillyMedia,2013.

[12] A.S.Tanenbaumand H.Bos,Modern OperatingSystems,4thed. UpperSaddle River, NJ,USA:Pearson, 2014.
[13] T.M.Mitchell,Machine Learning.NewYork,NY,USA:McGraw-Hill,1997.
[14] ISO/IEC25010,Systemsandsoftware engineering—Systemsand softwarequalityrequirementsandevaluation(SQuaRE)— System and software quality models, 2011.
[15] R.S.Pressman,Software Engineering:APractitioner's Approach,7thed.NewYork,NY,USA:McGraw-Hill, 2010.
[16] G.Coulouris,J.Dollimore,T.Kindberg,andG.Blair,DistributedSystems:Conceptsand Design,5th ed.Boston,MA,USA: Addison-Wesley, 2012.
[17] D.L.Parnas,"Onthecriteriatobeusedindecomposingsystemsintomodules,"CommunicationsoftheACM,vol.15,no.12,
[18] pp. 1053–1058,Dec.1972.
[19] D.Garlan,S.-W.Cheng,A.-C.Huang,B.Schmerl,andP.Steenkiste,"Rainbow:Architecture-basedself-adaptationwith
[20] reusable infrastructure," Computer,vol. 37,no.10, pp.46–54,Oct.2004.
[21] J.O.KephartandW.E.Walsh,"Anartificialintelligenceperspectiveonautonomiccomputingpolicies,"inProc.IEEEInt. Workshop Policies for Distributed Systems and Networks, 2004.
[22] P.Horn,"Autonomiccomputing:IBM's perspectiveonthe stateofinformationtechnology,"IBM Research,2001.
[23] N.Feamster,J.Rexford,andE.Zegura,"TheroadtoSDN:Anintellectualhistoryofprogrammablenetworks,"ACM
[24] SIGCOMMComputerCommunicationReview, vol.44, no. 2,pp. 87–98, 2014.
[25] R.SterrittandD.Bustard,"Autonomiccomputing—ameansofachievingdependability?,"inProc.IEEEInt.Conf.Engineering
[26] ofComplex ComputerSystems, 2003.
[27] B. H. C. Cheng et al., "Software engineering for self-adaptive systems: A research roadmap," in Software Engineering for Self- Adaptive Systems, Springer, 2009.
[28] Y. Brun et al., "Engineering self-adaptive systems through feedback loops," in Software Engineering for Self-Adaptive Systems,Springer,200



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)