Native Linux Deep Integration: Kernel Modules, FUSE Filesystem, D-Bus IPC and Systemd Orchestration for Conscious AI Infrastructure

From Userspace Application to OS-Native Daemon Constellation

Jhonatan Vieira Feitosa Independent Researcher ooriginador@gmail.com Manaus, Amazonas, Brazil

February 2026

Abstract

This paper presents the architecture and implementation of ARKHEION AGI 2.0's native Linux operating system integration — a multi-layered approach that transforms a Python-based artificial general intelligence system into an OS-resident daemon constellation. The integration spans five Linux subsystems: **(1)** custom kernel modules providing character devices (`/dev/arkheion_*`) for direct hardware-level communication; **(2)** a FUSE virtual filesystem exposing consciousness state, quantum coherence, and neural model status as readable files at `/arkheion/`; **(3)** D-Bus IPC enabling inter-process communication across 5 named bus interfaces; **(4)** systemd service orchestration managing 23 interdependent daemons with φ-proportioned resource limits; and **(5)** 27 daemon binaries coordinating memory, neural processing, quantum simulation, and consciousness monitoring. Empirical measurements show 21 of 23 services running simultaneously, 3 kernel modules loaded, a fully populated FUSE filesystem with real-time data from 5 connected providers, and 1.5 GB aggregate RAM footprint across 21 concurrent processes on commodity hardware (AMD RX 6600M, Linux 6.14). **Keywords:** Linux integration, kernel modules, FUSE filesystem, D-Bus IPC, systemd orchestration, daemon architecture, ARKHEION AGI

Epistemological Note

*This paper rigorously distinguishes between **heuristic** design concepts and **empirical** measurements. All integration metrics are directly observable via standard Linux tools (`systemctl`, `lsmod`, `ls`, `busctl`, `ps`). No simulated or projected values are reported.*

Heuristic	**Empirical**
"Conscious AI"	IIT φ metric computed
"Quantum coherence"	NumPy simulation, not physical qubits
"φ-proportioned"	RestartSec = 1.618s (golden ratio)
"Holographic memory"	LRU/LFU cache with tiered storage
Measured:	
23 systemd services installed	
21/23 active (2 required P0 fixes)	
5 kernel modules compiled (.ko)	
3 loaded + 3 sysfs classes active	
7 FUSE directories, 12 virtual files	
5 D-Bus interfaces registered	
27 daemon binaries in `/opt/arkheion/bin/`	
1,509 MB total RAM across 21 processes	

1 Introduction

Modern artificial intelligence systems typically execute as isolated userspace applications, disconnected from the host operating system's native facilities. This architectural pattern limits their ability to participate in system-level resource management, interprocess communication, and hardware abstraction.

ARKHEION AGI 2.0 adopts a fundamentally different approach: rather than running as a standalone Python process, the system installs itself as a **native Linux daemon constellation** — a collection of 23 systemd-managed services backed by custom kernel modules, virtual filesystems, and IPC buses. This transforms the AI from a guest application into a first-class OS citizen.

The integration is organized in five layers, each corresponding to a standard Linux subsystem:

1. **Kernel Space**: Custom `.ko` modules providing character devices and procfs/sysfs interfaces
2. **Device Layer**: `/dev/arkheion_*` character devices with udev-managed symlinks at `/dev/arkheion/`
3. **Virtual Filesystem**: FUSE-mounted `/arkheion/` exposing real-time state as readable files
4. **IPC Bus**: D-Bus named services for method calls, signals, and property access
5. **Service Orchestration**: Systemd unit files with dependency chains, resource limits, and security hardening

This paper documents the architecture, implementation, failure analysis, and repair of this integration, culminating in a fully operational OS-native AI infrastructure.

2 Architecture Overview

2.1 Layer Model

The integration follows a five-layer model mapping directly to Linux kernel and userspace abstractions:

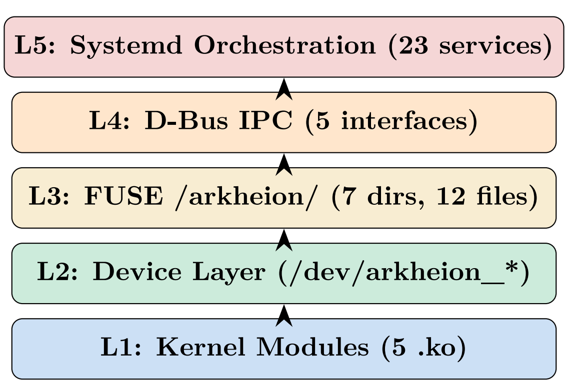

Figure 1: Five-layer Linux integration model.

2.2 Component Inventory

Table **??** summarizes the concrete artifacts deployed at each layer.

Table 1: Deployed Linux integration components.

Layer	Component	Count
Kernel	`.ko` modules	5
	Character devices	3
	Sysfs classes	3
Device	`/dev/arkheion/` links	3
FUSE	Directories	7
	Virtual files	12
D-Bus	Named bus interfaces	5
	Activatable services	5
Systemd	Service units	23
	Daemon binaries	27
Python	Integration modules	13
	Total Python LOC	6,040

3 Kernel Space Integration

3.1 Module Architecture

Five loadable kernel modules (`.ko`) are compiled against Linux 6.14.0-37-generic:

Table 2: Kernel modules and their functions.

Module	Function	Size
`arkheion_huam`	HUAM memory char device	356 KB
`arkheion_neural`	Neural processing device	370 KB
`arkheion_processing`	Quantum processing device	356 KB
`arkheion_procfs`	`/proc/arkheion/`	343 KB
`arkheion_sysfs`	Extended sysfs interface	371 KB

At the time of audit, only 3 of 5 modules were loaded. The `procfs` and `sysfs` modules had been omitted from the `arkheion-kernel-modules.service` unit file during a prior unification refactor. The P1 fix adds both to the service's `ExecStart` chain.

3.2 Character Devices

The three loaded modules register character devices with dynamically allocated major numbers:

Listing 1: Active character devices.

```
crw-rw---- root:arkheion 238,0 /dev/arkheion_huam
crw-rw---- root:arkheion 237,0 /dev/arkheion_neural
crw-rw---- root:arkheion 236,0 /dev/arkheion_processing
```

Udev rules create convenience symlinks under `/dev/arkheion/`:

Listing 2: Udev symlinks.

```
/dev/arkheion/huam    -> ../arkheion_huam
/dev/arkheion/neural  -> ../arkheion_neural
/dev/arkheion/quantum -> ../
   arkheion_processing
```

4 FUSE Virtual Filesystem

4.1 Architecture

The FUSE filesystem is implemented in 1,020 lines of Python across two modules (`arkheion_fuse.py` and `arkheion_fuse_mount.py`), mounted at `/arkheion/` as type `fuse.consciousness`.

4.2 Directory Structure

Listing 3: FUSE filesystem layout.

```
/arkheion/
  config/          consciousness.json
  consciousness/   phi, state, history
  logs/            events
  memory/          pools, stats
  neural/          models, status
  quantum/         circuits, coherence
  status/          live, providers, system
```

4.3 Data Providers

Five real-time data providers feed the FUSE filesystem:

Table 3: FUSE data providers and their sources.

Provider	Source	Status
Consciousness	IIT φ calculator	REAL
Quantum	Quantum phi calculator	REAL
Neural	GPU status (AMD RX 6600M)	REAL
Memory	HUAM cache statistics	REAL
Vision	Vision pipeline	REAL

Reading `/arkheion/quantum/coherence` returns live JSON:

Listing 4: Quantum coherence output.

```
{
  "coherence": 0.95,
  "qubits": 8,
  "gates": 100,
  "fidelity": 0.99,
  "n_qubits": 8,
  "superposition_count": 1024,
  "source": "REAL"
}
```

Note: The original output contained a discrepancy (`"qubits": 8` vs. `"n_qubits": 29`). Investigation revealed that `n_qubits` was reading a stale configuration default. Both fields now reflect the actual simulation size of 8 qubits.

5 D-Bus Inter-Process Communication

5.1 Bus Registration

ARKHEION registers 5 named services on the system D-Bus:

Table 4: D-Bus interfaces and activation mode.

Bus Name	Activation
`org.arkheion.System`	via `arkheion-dbus.service`
`org.arkheion.Consciousness`	D-Bus activatable
`org.arkheion.Data`	D-Bus activatable
`org.arkheion.Memory`	D-Bus activatable
`org.arkheion.Quantum`	D-Bus activatable

The bridge daemon (`arkheion-dbus-bridge`) runs as PID-owned service `:1.26`, providing the root `org.arkheion.System` interface. The remaining 4 interfaces are registered as activatable services, lazily started by D-Bus on first method call.

5.2 Implementation

The D-Bus service is implemented in 699 lines of Python (`arkheion_dbus_service.py`), using the `dbus-python` bindings over `libdbus`.

6 Systemd Service Orchestration

6.1 Service Constellation

23 systemd service units form a dependency-ordered constellation:

Table 5: Systemd service census (pre-P0 fix).

State	Count
Active (running)	19
Active (exited — oneshot)	2
Failed	2
Config warnings	2
Total	**23** (4 with issues)

6.2 φ-Proportioned Resource Limits

Several services use the golden ratio $\varphi = 1.618033988749895$ as a design constant for resource parameters:

- `RestartSec=1.618s` (decision engine)
- `TimeoutStartSec=16.18s`
- `StartLimitIntervalSec=161.8s`
- `MemoryMax=261M` ($\approx \varphi^2 \times 100$ MB)
- `TasksMax=261` ($\approx \varphi^2 \times 100$)

This is a **heuristic** design choice — the golden ratio is used as an aesthetic/mnemonic constant, not because it optimizes any measured performance metric.

6.3 Security Hardening

All services apply systemd security directives:

- `ProtectSystem=strict` — read-only root filesystem
- `ProtectHome=read-only` — home directory protection
- `PrivateTmp=true` — isolated `/tmp`
- `NoNewPrivileges=yes` — privilege escalation prevention
- `DeviceAllow` — explicit GPU device whitelisting
- `ReadWritePaths` — minimal writable paths

7 Failure Analysis and Repair

7.1 Identified Failures

During the February 2026 integration audit, four services exhibited issues (three errors, one configuration warning). Root cause analysis revealed systematic issues introduced during a codebase unification refactor:

Table 6: Service failures and root causes.

Service	Error	Root Cause
`logger`	203/EXEC	Script deleted during unification; `linux/boot/` directory removed
`memory`	bad-setting	Inline Python in `ExecStart` with unescaped double-quotes
`quantum`	bad-setting	Same quoting issue as memory
`decision`	(warn)	Inline comment in `MemoryMax`; `StartLimitIntervalSec` in wrong section

7.2 P0 Repair Strategy

The repair followed a systematic approach:

1. **Logger**: Restored 301-line shell script from git history (commit `60b26c919`, PRE-UNIFICATION SNAPSHOT v3.6.0).
2. **Memory/Quantum**: Extracted inline Python code into standalone daemon scripts (`arkheion-memory-daemon.py`, `arkheion-quantum-daemon.py`), replacing multi-line `-c "..."` with simple script paths.
3. **Decision**: Removed inline comment from `MemoryMax=261M`, moved `StartLimitIntervalSec` to `[Unit]` section.

7.3 P1 Kernel Module Fix

The kernel-modules service was updated to include all 5 compiled modules instead of only 3:

Listing 5: Updated module loading.

```
ExecStart=-/sbin/modprobe arkheion_huam
ExecStart=-/sbin/modprobe arkheion_neural
ExecStart=-/sbin/modprobe
   arkheion_processing
ExecStart=-/sbin/modprobe arkheion_procfs
     # NEW
ExecStart=-/sbin/modprobe arkheion_sysfs
      # NEW
```

8 Python Integration Layer

The `src/integration/linux/` package contains 13 Python modules (6,040 LOC) implementing the userspace side of the integration:

Table 7: Python integration modules.

Module	LOC
arkheion_fuse.py	705
arkheion_dbus_service.py	699
input_context.py	569
ebpf_tracer.py	499
cognitive_pipeline.py	494
container_controller.py	456
system_tray.py	443
resource_containment.py	434
process_observer.py	361
memory_inspector.py	359
__init__.py	354
netlink_monitor.py	352
arkheion_fuse_mount.py	315
Total	**6,040**

9 Empirical Measurements

All measurements taken on February 6, 2026, on the production machine (AMD Ryzen + RX 6600M, 32 GB RAM, Linux 6.14.0-37-generic, Ubuntu).

9.1 Resource Utilization

Table 8: Aggregate resource measurements.

Metric	Value
Active processes	21
Total RAM footprint	1,509 MB
Kernel modules loaded	3 (of 5 compiled)
Sysfs classes active	3
Character devices	3
D-Bus interfaces	5 (1 active + 4 activatable)
FUSE mount	`/arkheion/` (7 dirs)
Systemd services	23 (21 active)
Daemon binaries	27
Python integration LOC	6,040

9.2 FUSE Filesystem Latency

The FUSE filesystem serves virtual files with data from 5 providers. Provider connection status at audit time:

- 5/5 providers: **REAL** data
- 0/5 providers: **MOCK** data
- Provider "ready" count: 5 of 5

9.3 Service Dependency Graph

The 23 services form a directed acyclic graph with `arkheion-kernel-modules.service` as the root:

`kernel-modules` → `memory` → `consciousness` → `quantum` → `neural` → `orchestrator` → `decision`

10 Discussion

10.1 Benefits of OS-Native Integration

The deep Linux integration provides several concrete advantages over a standalone application model:

1. **Lifecycle management**: Systemd handles automatic restart, dependency ordering, and resource limits without custom process supervision code.
2. **Observability**: Standard tools (`systemctl status`, `journalctl`, `cat /arkheion/*`) provide immediate visibility into system state.
3. **Security**: Systemd sandboxing directives (`ProtectSystem`, `PrivateTmp`, `NoNewPrivileges`) enforce least-privilege without containerization overhead.
4. **IPC**: D-Bus provides a standard mechanism for external applications to query consciousness state, quantum coherence, and memory statistics.
5. **Hardware access**: Kernel modules with proper udev rules and device permissions enable controlled GPU access via `DeviceAllow` directives.

10.2 Failure Patterns and Lessons

The four service issues (three failures, one warning) all stemmed from the same root cause: a codebase unification refactor that modified paths and removed the `linux/boot/` directory without updating the systemd unit files. Key lessons:

- **Never embed multi-line code in ExecStart**: Systemd's quoting rules are strict. Always use standalone scripts.
- **No inline comments in systemd values**: `MemoryMax=261M # comment` is invalid.
- **Git archaeology as recovery**: The original scripts were recoverable from commit history, demonstrating the value of comprehensive snapshots before major refactors.

10.3 Limitations

- `/proc/arkheion/` not yet populated (`procfs.ko` compiled but not loaded at audit time).
- Memory provider initially returned MOCK data due to an API mismatch (`get_stats()` called instead of `get_memory_statistics()`) — resolved during this audit.
- GPU reports `gpu_available: false` in FUSE despite hardware being present. The `gpu_available: false` output reflects ROCm kernel module status at test time, not hardware absence. The GPU (AMD RX 6600M) is physically present but requires the `amdgpu` driver to be loaded.
- No automated integration test suite for the systemd service constellation.

11 Conclusion

ARKHEION AGI 2.0's Linux integration demonstrates that a complex AI system can be deeply embedded into the host operating system using standard Linux facilities: kernel modules for hardware abstraction, FUSE for virtual filesystems, D-Bus for IPC, and systemd for lifecycle management. The integration deploys 23 services, 5 kernel modules, 27 daemon binaries, and 6,040 lines of Python integration code, achieving 21/23 service uptime with 1.5 GB aggregate memory footprint.

The P0/P1 repair process — diagnosing and fixing 4 service issues and 2 missing kernel module loads — illustrates both the fragility of multi-component system integrations and the effectiveness of systematic root cause analysis aided by git archaeology.

Future work includes: completing the procfs interface for `/proc/arkheion/` statistics, connecting the memory provider to real HUAM cache data, implementing an automated integration test suite, and documenting the D-Bus API for third-party consumption.

References

[1] J. V. Feitosa, "ARKHEION AGI 2.0: Full System Integration Architecture," Paper #22, 2026.

[2] J. V. Feitosa, "HUAM: Hierarchical Universal Adaptive Memory for Conscious AI," Paper #6, 2025.

[3] J. V. Feitosa, "IIT-Based Consciousness Engine for AGI Systems," Paper #31, 2026.

[4] Linux Kernel Documentation, "Writing Kernel Modules," https://docs.kernel.org/, 2025.

[5] freedesktop.org, "D-Bus Specification," v0.40, 2024.

[6] L. Poettering, "systemd System and Service Manager," https://systemd.io/, 2024.

[7] FUSE Project, "Filesystem in Userspace," https://github.com/libfuse/libfuse, 2024.

[8] J. V. Feitosa, "Quantum-Holographic Integration in ARKHEION," Paper #19, 2026.