Sistema Nemosine Nous — UML Overview v0.1

Arquitetura Mínima Executável (AME)

1. Objetivo

Este documento apresenta a **Arquitetura Mínima Executável (AME)** do **Sistema Nemosine Nous**, descrita por meio de um conjunto completo e coerente de **diagramas UML**.

O objetivo da AME é explicitar o **ciclo mínimo funcional de interação humano–IA**, garantindo clareza arquitetural, auditabilidade, separação de responsabilidades e governança do comportamento do sistema, sem antecipar decisões de implementação desnecessárias.

2. Visão Geral da Arquitetura

O Sistema Nemosine Nous é estruturado em torno de um **Orquestrador AME**, responsável por coordenar o ciclo de execução entre:

- a **Interface**, que atua como borda de entrada e saída do sistema;
- a **Memória**, responsável pelo estado persistente e contextual;
- um **LLM externo**, tratado como dependência encapsulada;
- e mecanismos de **controle comportamental externo (Overtuning)**.

A arquitetura privilegia controle explícito de fluxo, rastreabilidade e desacoplamento, permitindo evolução incremental sem dependência de fine-tuning do modelo de linguagem.

3. Diagramas UML Incluídos

O conjunto UML documenta o sistema sob múltiplas perspectivas complementares:

- **Diagrama de Casos de Uso** — interação entre atores e o ciclo AME
- **Diagrama de Atividades** — fluxo operacional do ciclo mínimo
- **Diagrama de Sequência** — interação temporal entre Interface, Orquestrador, Memória e LLM (happy path e falhas)
- **Diagrama de Estados** — estados internos relevantes do ciclo AME
- **Diagrama de Componentes** — responsabilidades funcionais e dependências lógicas
- **Diagrama de Implantação** — visão física/lógica de execução e comunicação
- **Diagrama de Pacotes** — organização modular do código e responsabilidades
- **Diagrama de Classes** — estrutura conceitual das principais entidades do núcleo AME
- **Diagrama de Objetos** — instância concreta de um ciclo de execução em um momento específico

Em conjunto, os diagramas formam uma visão arquitetural completa, consistente e auditável.

4. Princípio de Overtuning

O sistema adota o princípio de **Overtuning**, entendido como:

Controle externo do comportamento de modelos de linguagem por meio de arquitetura, orquestração e regras explícitas, sem modificação dos pesos internos do modelo (fine-tuning).

O Overtuning é tratado como **abordagem transversal**, não como componente executável, e se manifesta por meio de decisões arquiteturais, regras de orquestração e validações de fluxo.

Metaprompt (Artefato Central de Controle Cognitivo) - O Sistema Nemosine Nous opera sob um metaprompt estruturante, responsável por impor regras, papéis, restrições e diretivas de estado ao LLM, viabilizando a abordagem de *Overtuning* — controle externo de comportamento sem fine-tuning do modelo.

5. Escopo e Limitações

Este conjunto UML descreve:

- a arquitetura lógica e estrutural do sistema;
- o ciclo mínimo funcional (AME);
- as responsabilidades e dependências entre módulos

Deliberadamente **não cobre**:

- detalhes de implementação em nível de código;
- escolhas finais de frameworks, bancos ou provedores;
- métricas de desempenho ou escalabilidade.

Esses aspectos são tratados em artefatos técnicos posteriores.

6. Propriedade Intelectual e Registro

Registro de Programa de Computador (INPI – Brasil) - O Sistema Nemosine Nous encontra-se formalmente registrado junto ao Instituto Nacional da Propriedade Industrial (INPI), sob o número: **BR512025003335-4**, garantindo proteção autoral sobre a implementação e a estrutura funcional do sistema.

A publicação deste conjunto UML tem caráter **técnico, documental e de rastreabilidade**, não implicando renúncia a direitos autorais ou de propriedade intelectual.

7. Status do Documento

- **Estado:** Consolidado
- **Nível:** Arquitetura mínima executável (AME)
- **Uso previsto:** Documentação técnica, auditoria arquitetural, base para implementação incremental

Diagramas UML do Sistema Cognitivo Modular Vivo Nemosine Nous © v0.1

Diagrama de Casos de Uso (Use Case Diagram)

Mostra **quem pode fazer o quê** no sistema. Define atores, capacidades e fronteiras funcionais. Responde: *"Quais funcionalidades existem e quem as aciona?"*

Diagrama de Classes (Class Diagram)

Mostra a **estrutura estática** do sistema. Classes, atributos, métodos e relações. Responde: *"Quais são os blocos conceituais do sistema?"*

Diagrama de Implantação (Deployment Diagram)

Mostra **onde o sistema roda fisicamente/logicamente**. Servidores, APIs, serviços externos, LLMs. Responde: *"Em que infraestrutura isso vive?"*

Diagrama de Atividades (Activity Diagram)

Mostra **o fluxo passo a passo** de uma execução. Inclui sequência, decisões, loops e paralelismo. Responde: *"O que acontece, em que ordem, quando o UC03 roda?"*

Diagrama de Estados (State Machine Diagram)

Mostra os **estados possíveis** de algo (ex: Sessão). E as transições entre eles. Responde: *"Em que estados o sistema pode estar?"*

Diagrama de Objetos (Object Diagram)

Mostra um **snapshot concreto** de instâncias em execução. Objetos reais em um momento específico. Responde: *"Como isso fica em tempo de execução?"*

Diagrama de Sequência (Sequence Diagram)

Mostra **quem chama quem, e quando**, ao longo do tempo. Explicita mensagens entre User, Orchestrator, LLM, Memória. Responde: *"Como as partes conversam durante o ciclo?"*

Diagrama de Componentes (Component Diagram)

Mostra os **módulos técnicos** e suas dependências. Backend, Orchestrator, Adapter LLM, Memory Store etc. Responde: *"Como o sistema é modularizado?"*

Diagrama de Pacotes (Package Diagram)

Organiza o sistema em **grandes blocos conceituais**. Agrupa classes, componentes e responsabilidades. Responde: *"Como o conhecimento do sistema é organizado?"*

Diagrama de Casos de Uso (Use Case Diagram)

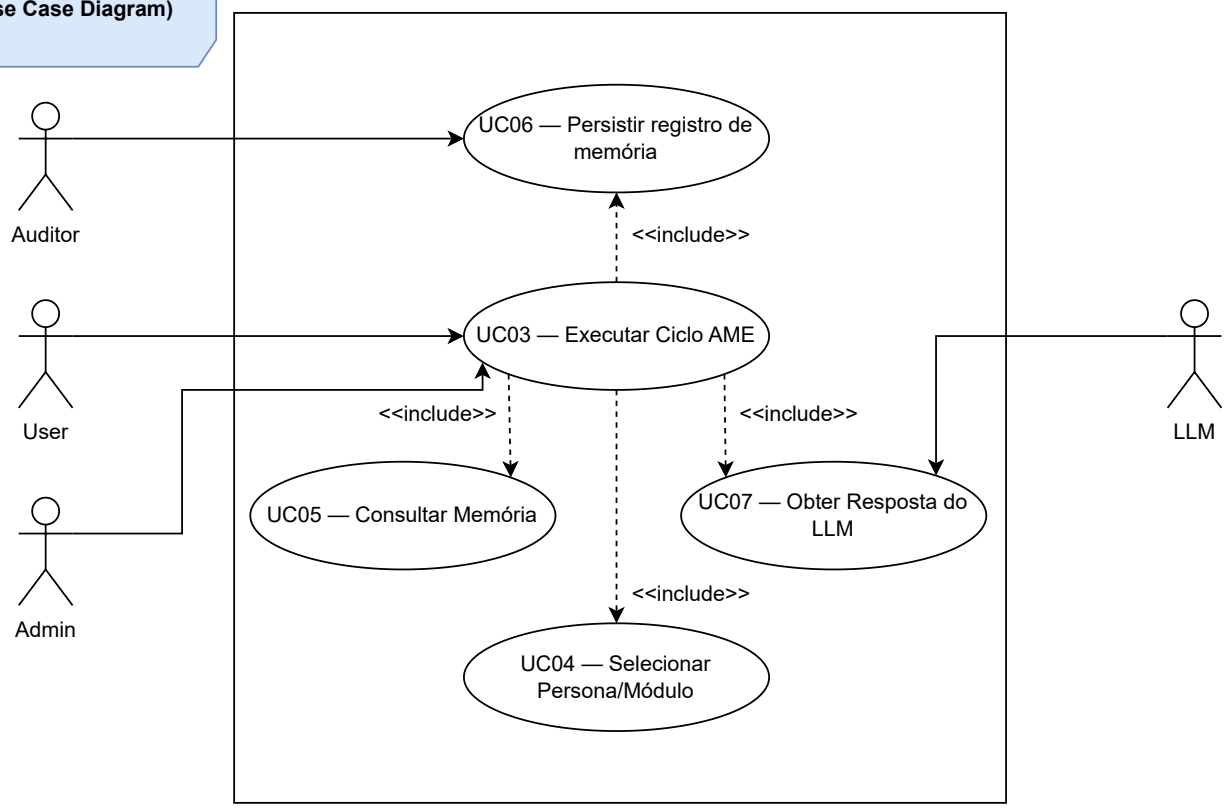


Diagrama de Atividades (Activity Diagram)

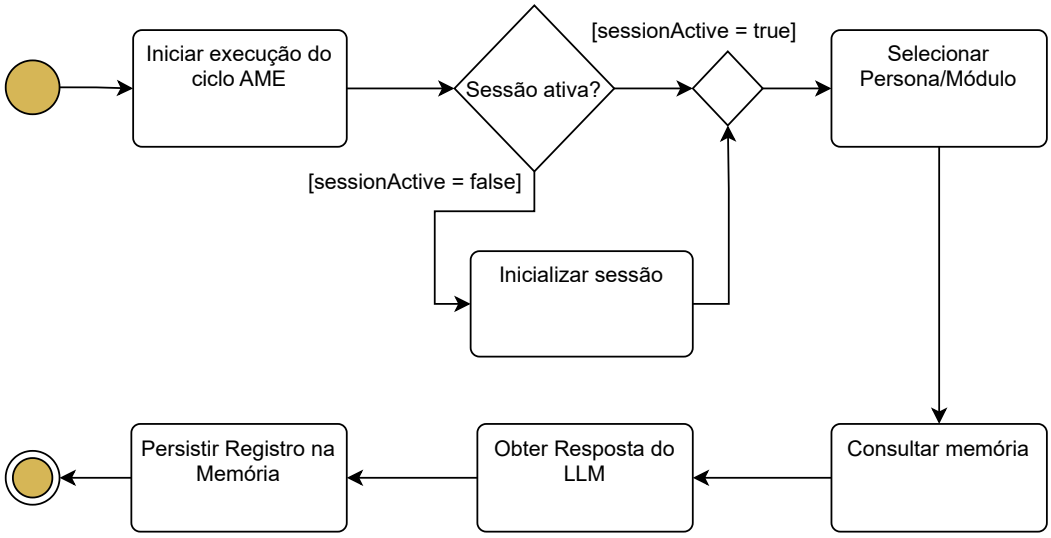


Diagrama de Sequência — Ciclo AME (Happy Path)

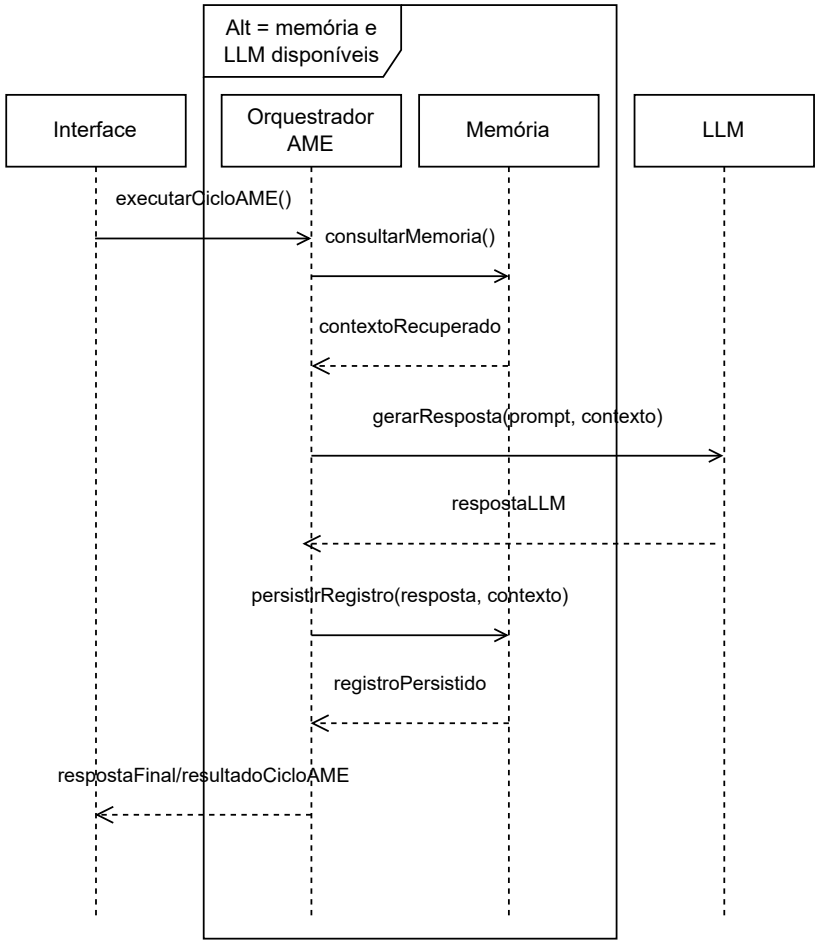


Diagrama de Classes (Class Diagram)

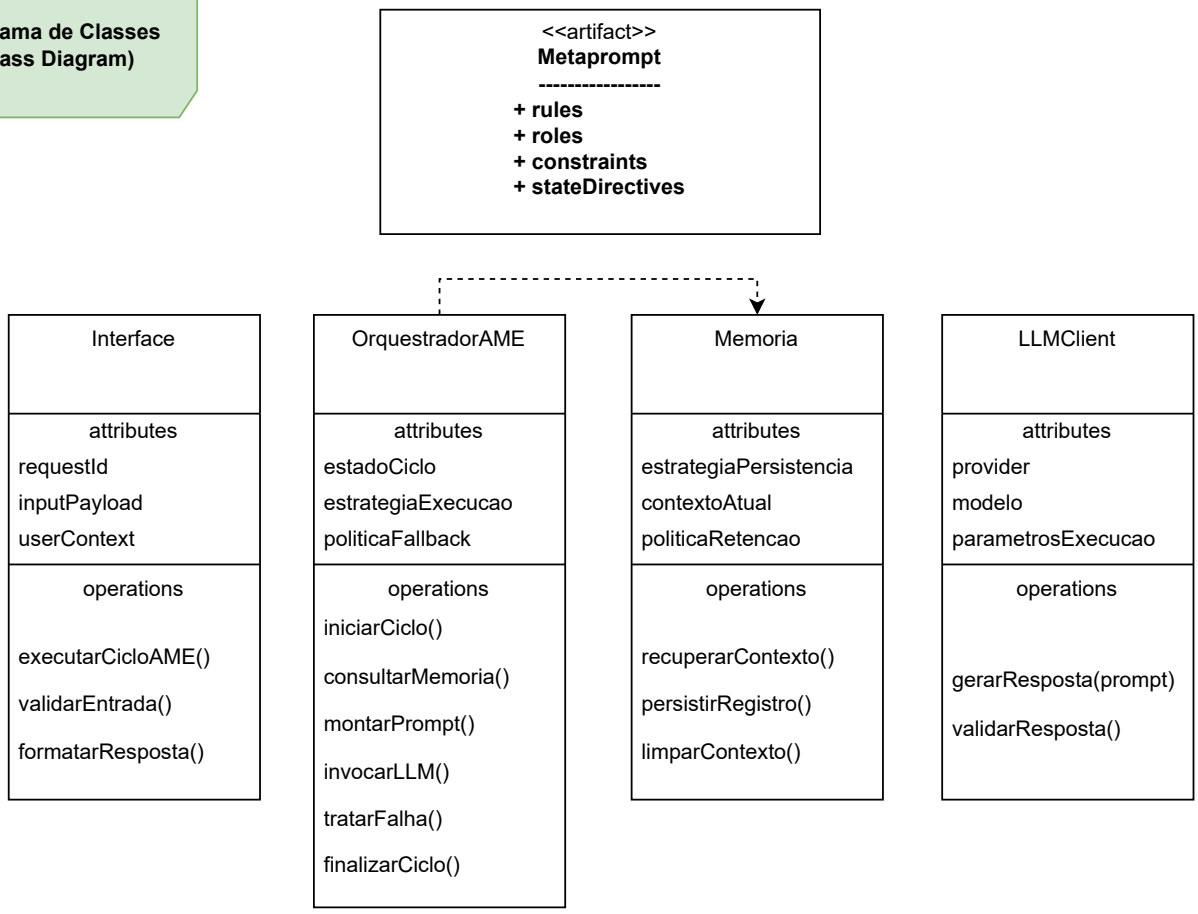


Diagrama de Estados (State Machine Diagram)

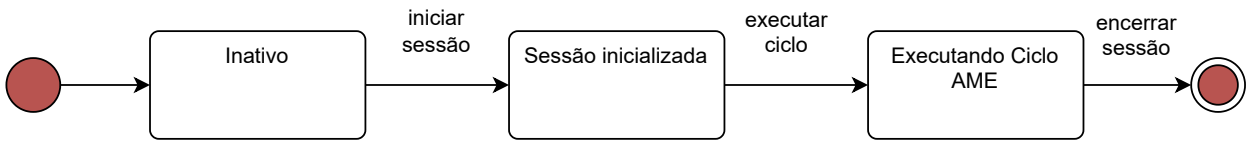


Diagrama de Componentes (Component Diagram)

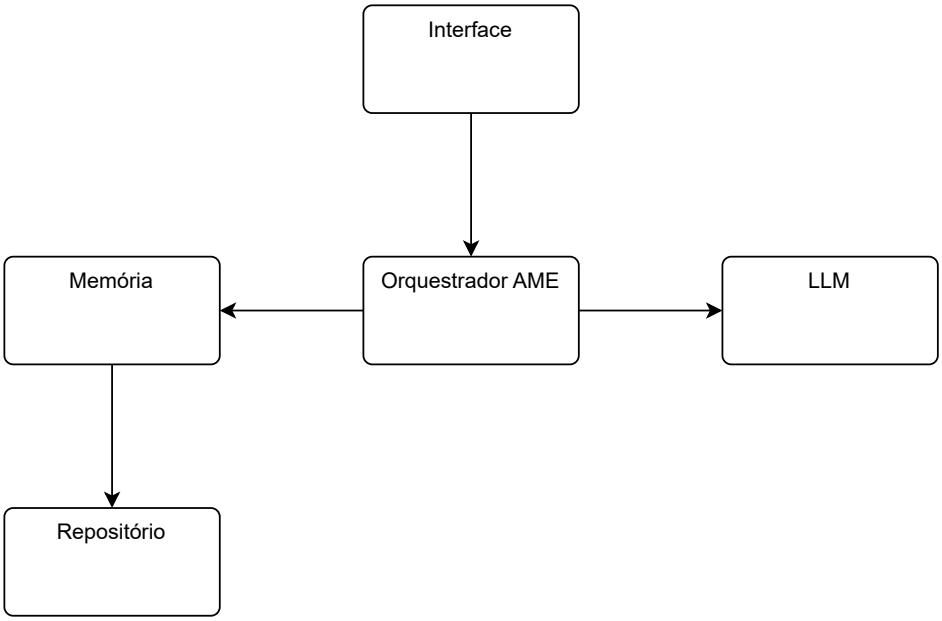


Diagrama de Implantação – Stack Tecnológico do AME

«approach»
OVERTUNING
External behavior control without model fine-tuning

Banco de Dados
(Memória Persistente)

- PostgreSQL *(ou SQLite, se preferir MVP)*
- SQLAlchemy
- Tabela: Contextos / Registros AME

Servidor de Aplicação
(Backend AME)

- Python 3.x
- FastAPI *(ou Flask, a escolher)*
- Orquestrador AME
- Camada de Overtuning (Prompt Control)

Cliente / Navegador Web

- HTML + CSS + JS *(ou React)*

LLM Externo

- OpenAI API *(ou outro, se quiser)*
- Comunicação: HTTPS / REST
- Sem fine-tuning (Overtuning externo)

Diagrama de Objetos (Object Diagram)

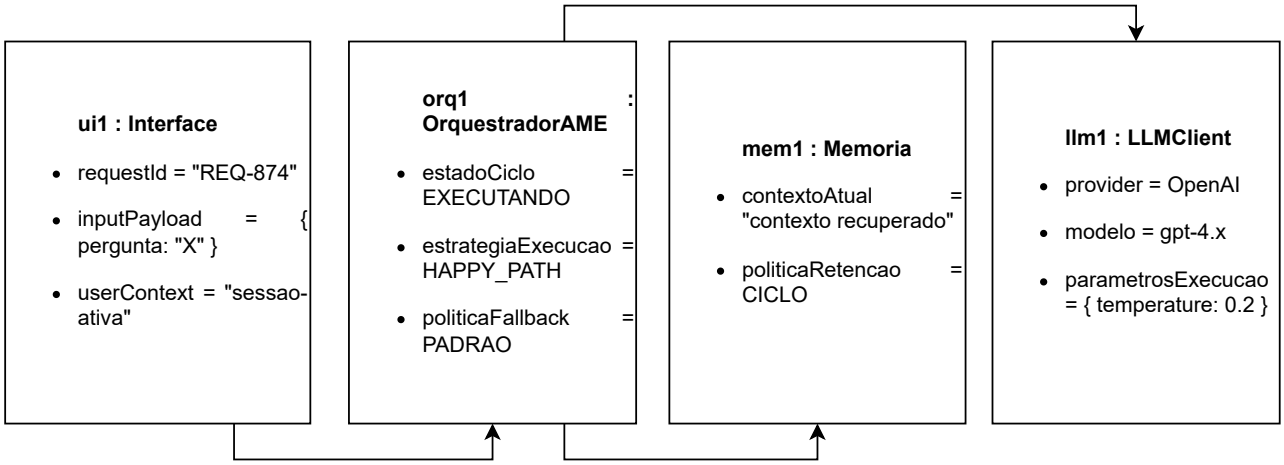
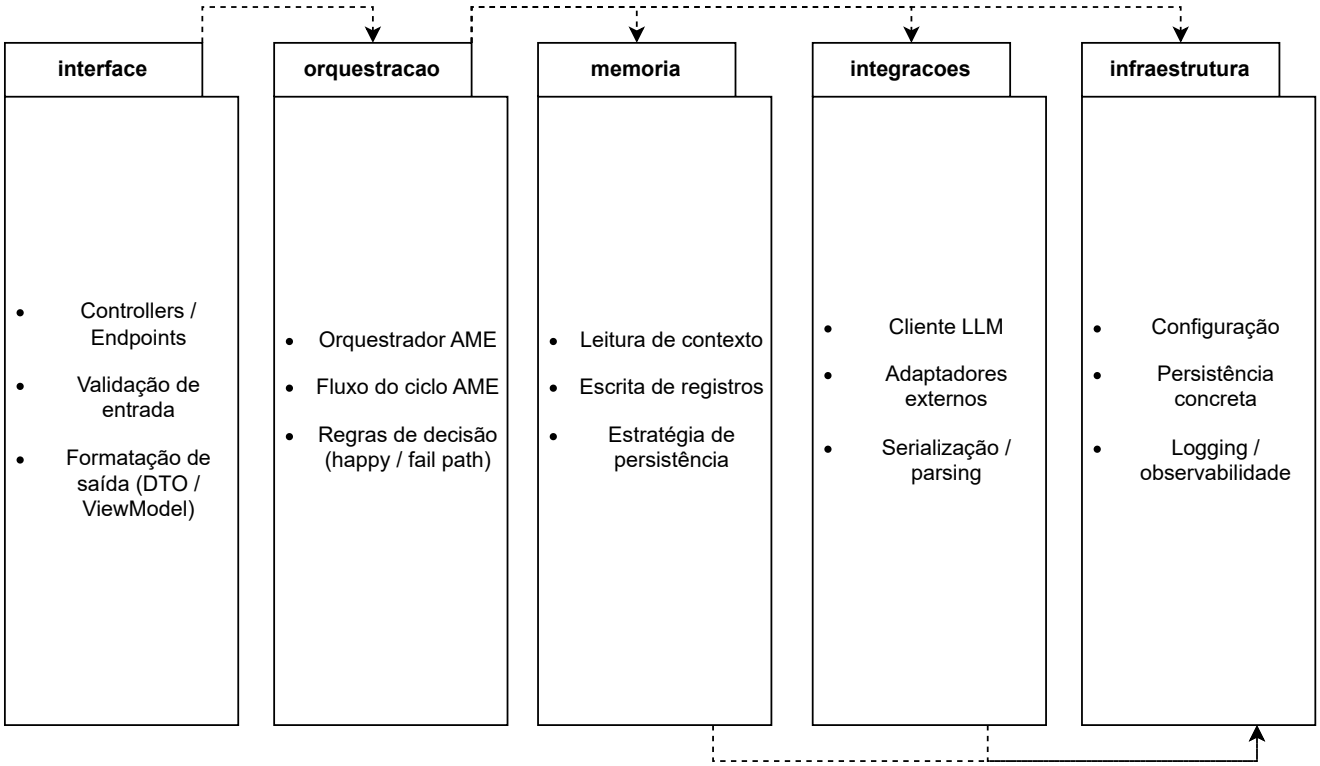
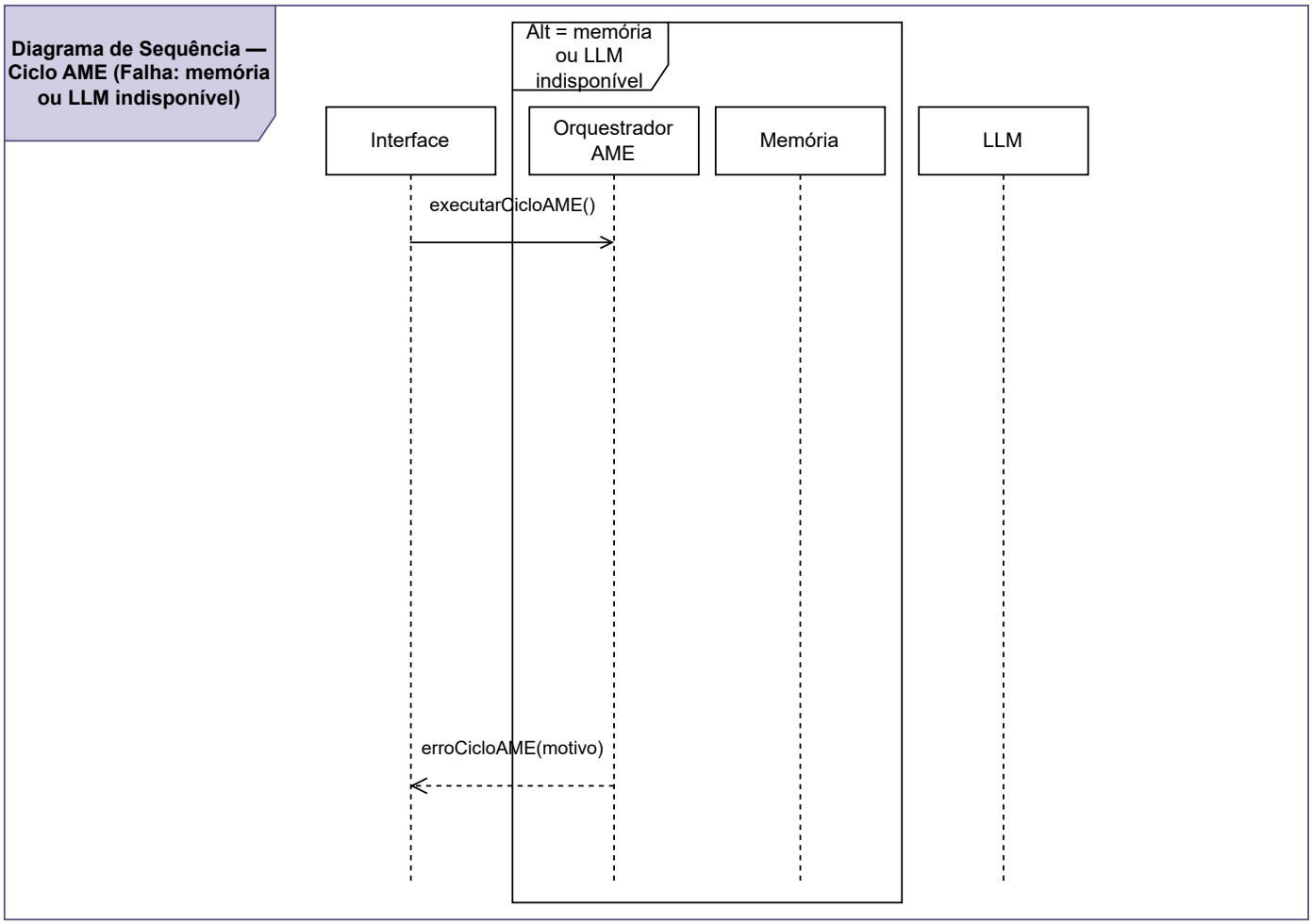


Diagrama de Pacotes (Package Diagram)





Links úteis

Zenodo Community (official archival repository): https://zenodo.org/communities/sistema-nemosine/

OSF (process-level artifacts and development records): https://osf.io/r4yf8/overview?view_only=87d0f8a36ada4b5baa4f6361d52c7bd8

Onboarding and project entry point: https://zenodo.org/records/18072732

GitHub (structural, legal, and version control records): https://github.com/edersouzamelo

Linktree (navigation hub): https://linktr.ee/NemosineNous

ORCID (author profile): https://orcid.org/0009-0003-6835-135X

LinkedIn (author profile): https://linkedin.com/in/edersouzamelo/