13 — Interfaces

🕒 estado da arte 2026-07 · revisão 2026-07-26 · 📖 ~13 min de leitura

Objetivos de aprendizagem

Ao final deste capítulo, você deve ser capaz de:

1. **Argumentar** por que "núcleo com front-ends" bate "front-end com um agente dentro" — e como desenhar a fronteira cedo maximiza as superfícies possíveis;
2. **Distinguir** as superfícies (TUI (Terminal User Interface), headless/SDK (Software Development Kit), IDE (Integrated Development Environment), chat, cloud) e o que cada uma exige do core;
3. **Avaliar** a UX de interação à luz da HCI (Human-Computer Interaction) (mixed-initiative, níveis de automação, over-reliance);
4. **Reconhecer** a superfície como fronteira de segurança (mesmo contrato de turn, não backdoor) e a virada para o paradigma *ambient*/inbox;
5. **Explicar** por que, com o loop atrás de portas, uma segunda superfície (headless) é um adapter fino, não uma reescrita (etapa 0 do harness-zero).

O problema

O mesmo agente precisa servir públicos diferentes: o desenvolvedor no terminal, o script de CI que precisa de JSON, o IDE que quer diffs inline, o gestor que acompanha por chat. A pergunta arquitetural é uma só: **o harness é um núcleo com múltiplos front-ends, ou um front-end com um agente dentro?** Os harnesses estudados responderam "núcleo com front-ends" — e a qualidade dessa separação determina quantas interfaces são viáveis.

Superfícies consagradas: **TUI interativa**, **headless/não-interativo** (`-p` com saída estruturada), **IDE** (diffs, contexto do editor), **CI/CD** (Actions), **protocolos de agente** (ACP (Agent Client Protocol), A2A (Agent-to-Agent)), **chat** (Slack, Telegram…) e, cada vez mais, **cloud/assíncrona**.

Fundamentos científicos

Registro editorial honesto (Princípio I): **não existe canon acadêmico de "interface de harness de agente"** — a lacuna é real. Mas a HCI de interação humano-IA a fundamenta com precisão, e um filete recente (2025-26) já trata de human-in-the-loop de agentes.

- **Quando agir × quando perguntar** — Principles of Mixed-Initiative UI (Horvitz, CHI '99): os 12 princípios sobre incerteza do objetivo, custo/benefício de agir e handoff gracioso *são* a decisão central de um harness — plan mode e aprovações (caps. 07/09) são "passar a iniciativa" aplicado.

- **O dial de autonomia é por estágio** — a escala de 10 níveis de automação (Sheridan & Verplank, 1978) e o modelo de tipos e níveis (Parasuraman, Sheridan, Wickens, 2000) mostram que a automação se aplica *independentemente* a cada estágio (aquisição · análise · decisão · ação). Decisão: o harness pode **auto-coletar contexto** (automação alta) e ainda **gatear a ação** (automação baixa) — o dial não precisa ser global.
- **A UX de "quando errar"** — Guidelines for Human-AI Interaction (Amershi et al., CHI '19): 18 diretrizes por fase; as de recuperação (correção/desfazer barato) explicam por que a reversibilidade (cap. 08) é também uma decisão de *interface*.
- **A supervisão é frágil — projete contra isso** — To Trust or to Think (Buçinca et al., CSCW '21) mostra que explicação sozinha **não** cura over-reliance; *forcing functions* cognitivas sim. A revisão de over-reliance (Passi & Vorvoreanu, MSR-TR-2022-12) sintetiza o risco. Decisão: a aprovação deve ser um **ato deliberado**, não um clique reflexo, e a superfície não pode esconder o que o agente fez. O trabalho recente de human-in-the-loop de agentes (Magentic-UI, arXiv 2507.22358, com *action guards* = gating de permissão; design de oversight, arXiv 2510.19512) operacionaliza isso.

(Bibliografia completa e ponteiros: `livro/bibliografia.md`.)

Fontes da indústria

- **Um núcleo, muitas superfícies (agora doutrina)** — a doc Platforms and integrations (Claude Code) diz explicitamente: "roda o mesmo motor subjacente em todo lugar, mas cada superfície é afinada para um jeito de trabalhar" (CLI, Desktop, VS Code, JetBrains, Web, Mobile + Chrome, GitHub Actions, GitLab, Slack), com **config, memória de projeto e MCP (Model Context Protocol) compartilhados** entre as superfícies locais. Decisão: construa o agente como um motor único e trate terminal/IDE/web/mobile como front-ends intercambiáveis.
- **Headless é um filtro Unix** — Run Claude Code programmatically (headless): `-p`/`--print`, `--output-format text|json|stream-json`, lê stdin e redireciona stdout "como qualquer ferramenta de linha de comando", com `--allowedTools`/`--permission-mode` para runs desatendidos nunca travarem num prompt. Decisão: a interface é stdin/stdout + exit codes — o agente cai em pipes, build scripts e CI sem UI.
- **O SDK é o loop empacotado; Managed Agents é o agente como serviço** — o Agent SDK dá "as mesmas tools, loop e gestão de contexto que movem o Claude Code", programável em Python/TS, e separa *quem roda o loop* (SDK no seu processo) de *quem o renderiza*; os Managed Agents levam ao extremo — "a Anthropic roda o agente e o sandbox, sua aplicação manda eventos e recebe o stream". Decisão: a superfície programática é o core como biblioteca — ou como endpoint REST.
- **O IDE é uma superfície fina sobre o mesmo motor** — VS Code e JetBrains adicionam diffs inline e contexto do editor reusando o engine do CLI (mesmo CLAUDE.md, mesmos modos de permissão); o padrão mais amplo — Copilot agent mode, Cursor 2.0 (background agents), Windsurf Cascade — divide a superfície do editor em **inline (síncrona)** e **background (assíncrona, cloud)** sobre a mesma abstração de tarefa.
- **A UX de interação: aprovação como máquina de estados, streaming como evento, humano como tool** — os modos de permissão fazem da aprovação uma máquina de estados

(default/acceptEdits/plan/…), não um prompt ad-hoc; o streaming expõe o loop como stream tipado de eventos (`text_delta`, `tool_use`, `result`); e o AskUserQuestion modela o human-in-the-loop **como uma tool** que o agente chama — "perguntar ao humano" vira um passo do loop, não uma interrupção especial.

- **A virada ambient/inbox** — para agentes assíncronos, a superfície deixa de ser o prompt de chat e vira uma **caixa de entrada**: os ambient agents da LangChain (sempre-ligados, disparados por evento, que emergem ao humano só por notify/question/review) e o Claude Code na web (roda em cloud gerenciada e "continua depois que você desconecta") apontam o mesmo futuro: supervisionar *muitos* agentes de longa duração sem um terminal ao vivo — exatamente o "oversight sem oversight constante" que a HCI de níveis de automação prevê.
- **Chat como serviço, com identidade própria** — os channels deixam Telegram/Discord "ou seu próprio servidor" empurrar eventos para uma sessão; o Slack faz `@Claude` virar sessão cloud que transforma bug em PR, **com credenciais e trilha de auditoria próprias, desacopladas do acesso de qualquer humano**. Decisão: chat é só mais um gatilho, e o agente-serviço tem identidade própria (liga ao cap. 07).
- **Consulte também**: a coleção viva Awesome Harness Engineering — Human-in-the-Loop reúne mais recursos consultáveis desta dimensão (padrões, artigos e implementações), curados por problema.

O estado da arte

1. Núcleo com front-ends — a fronteira cedo decide tudo

A lição estrutural da rodada 1 virou consenso: **quanto mais cedo a fronteira núcleo/interface é desenhada, mais interfaces cabem depois**. O Codex é o exemplo cristalino — "um único motor Rust serve TUI, `codex exec` headless, extensão IDE, app desktop, cloud/web, servidor MCP e remote control". O opencode paga a mesma aposta com uma API HTTP tipada e clientes gerados. O anti-padrão é o inverso: um front-end com um agente enfiado dentro, que não escala para uma segunda superfície sem reescrita.

2. Headless com saída estruturada é obrigatório

Não há harness sério sem o modo filtro-Unix: `codex exec` (JSONL), gemini `--output-format stream-json` (NDJSON de eventos), `oh -p`/`ohmo --print`, Aider headless. A saída estruturada é o que torna o agente **programável** — peça de pipeline, alvo de CI, backend de outra UI. É a superfície que, uma vez ausente, fecha todas as outras automações.

3. Três visões — e a explosão do "colega no chat" com voz

As três apostas da rodada 1 persistem, agora mais nítidas: o agente como **produto** multi-plataforma (opencode Electron/VS Code; Codex desktop+cloud), como **serviço** de plataforma (gemini-cli SDK/A2A/Action; Managed Agents REST) e como **colega** no chat. A categoria de agentes pessoais *explodiu* a terceira: o **OpenClaw** serve **~23 canais** de chat + apps nativos (iOS/Android/macOS/Windows) + **voz** (Voice Wake, Talk Mode contínuo) + Live Canvas; o **Hermes** tem um gateway multi-canal de processo único (10 plataformas + voz); o **ohmo** faz de Telegram/Slack/Discord/Feishu a superfície primária. A voz e a largura de canais viraram superfícies de primeira classe.

4. A superfície é fronteira de segurança, não backdoor

A lição mais madura da rodada 2, e a que a HCI de over-reliance reforça: uma superfície não pode ser um atalho em volta do core. O **IronClaw** materializa isso — CLI, WebUI, Slack, Telegram e webhooks entram todos pelo **mesmo contrato de turn** (`ProductAdapter`), e a WebUI é *proibida* de bypassar as fronteiras de auth. O agente-no-Slack roda com credenciais e auditoria próprias, desacopladas do humano. E a UX precisa não *esconder* o que o agente fez — o antídoto contra a falsa sensação de supervisão que Buçinca e Passi & Vorvoreanu documentam.

5. A próxima fronteira: ambient, cloud, assíncrono

O paradigma emergente muda a própria natureza da interface. O **Codex cloud-tasks** (TUI de tarefas remotas), o Claude Code na web que continua após desconectar, e o inbox dos ambient agents apontam para o mesmo lugar: o humano deixa de *dirigir* um agente ao vivo e passa a *supervisionar muitos* por notificação e revisão. É o dial de autonomia da HCI levado ao produto — automação alta na execução, o humano no gate de decisão, assíncrono. A interface do agente está saindo do terminal (o watch mode do Aider transforma comentários `ai!` em qualquer editor; o Live Canvas do OpenClaw) e virando ambiente.

> **LEITURA EXECUTIVA**
>
> O que está mais moderno: um motor, muitas superfícies (doutrina); headless estruturado obrigatório; a explosão de canais + voz; a superfície como fronteira de segurança (mesmo contrato de turn); e a virada ambient/inbox/cloud. **O que roubar:** desenhe a fronteira núcleo/interface cedo (API tipada ou biblioteca), não tarde; entregue headless com `stream-json` desde o dia um; faça toda superfície passar pelo mesmo contrato de turn (nunca um backdoor de auth); modele o human-in-the-loop como tool e a aprovação como ato deliberado; e prepare-se para o inbox — o próximo terminal é assíncrono.

Mão na massa — harness-zero: o chat como janela de observação

A interface do harness-zero nasceu na **etapa 0**: um chat mínimo sobre FastAPI, a *janela de observação* que acompanha cada etapa do livro. A lição desta dimensão é o que o projeto inteiro demonstra: porque o loop vive atrás de portas (`LLMPort`, `ToolPort`, `StorePort`), acrescentar uma **segunda superfície** — um modo headless `--print` que emite os mesmos eventos em `stream-json` — é um **adapter fino**, não uma reescrita. Exercício de completude: você adiciona o modo headless e prova que o mesmo agente responde no chat e no pipe, e que a aprovação (o gate de permissão do cap. 07) aparece nas duas superfícies pelo mesmo contrato — a superfície não é backdoor.

Verificação

1. Por que "núcleo com front-ends" permite mais interfaces do que "front-end com um agente dentro", e o que decide isso na prática? (A fronteira desenhada cedo — API tipada/biblioteca — deixa cada superfície

ser um adapter fino; o inverso exige reescrita por superfície.)

2. Seu agente vai rodar assíncrono em cloud, supervisionado por vários humanos. Que paradigma de interface e que princípio de HCI guiam o design? (Ambient/inbox — notify/question/review; níveis de automação: alta na execução, humano no gate de decisão; over-reliance → não esconder o que o agente fez.)
3. Você expõe o agente no Slack e numa WebUI. Que regra impede que a nova superfície vire um furo de segurança? (Mesmo contrato de turn/ProductAdapter — toda superfície passa pelas mesmas fronteiras de auth; a UI não bypassa; identidade/auditoria próprias.)

Apêndice A — Como cada repositório trata as interfaces

> Evidência por harness, com paths — complementação online, expandida a cada rodada.

opencode (rodada 1) — a maior superfície de produto

Arquitetura cliente-servidor (cap. 02): servidor HTTP com API tipada e clientes gerados habilitam **sete superfícies** — TUI (SolidJS/opentui), **app desktop Electron** (única na rodada 1), extensão VS Code, **GitHub Action** (`packages/github/`), **Slack** (`packages/slack/`), web (`packages/web/`) e **ACP** (integração Zed). Sessões compartilháveis por link conectam as superfícies.

gemini-cli (rodada 1) — terminal rico + plataforma

TUI React/Ink com ~40 slash commands por loaders plugáveis. **Headless de primeira classe**: `gemini -p` com `--output-format stream-json` (NDJSON em tempo real). **VS Code companion** (servidor IDE expondo arquivos/diffs). GitHub Action oficial. **ACP** para editores e **A2A server**. SDK próprio (`packages/sdk`).

OpenHarness/ohmo (rodada 1) — o agente que mora no chat

CLI Typer (`oh`) headless (`-p`, `text|json|stream-json`) + `--dry-run`; duas TUIs (React/Ink + Textual); dashboard web do autopilot. E o **ohmo**: agente pessoal em **Telegram/Slack/Discord/Feishu** (`channels/` + `gateway/`) com workspace próprio.

OpenClaw (rodada 2) ⭐ — a maior largura de superfície

~23 canais (WhatsApp, Telegram, Slack, Discord, Signal, iMessage, Teams, Matrix, Feishu, LINE, WeChat, QQ…), Control UI web, WebChat, CLI, TUI, **voz** (Voice Wake + Talk Mode contínuo), **apps nativos** (iOS/Android/macOS/Windows) e **Live Canvas** (A2UI). A superfície do agente como produto de consumo.

Codex CLI (rodada 2) ⭐ — um motor, todas as superfícies

Um único motor Rust serve: TUI (ratatui), `codex exec` headless (humano + JSONL), extensão IDE via App Server, **app desktop**, **cloud/web** (`cloud-tasks` com TUI de tarefas remotas), Codex como servidor MCP e remote control. O exemplo canônico de núcleo com front-ends.

IronClaw (rodada 2) ⭐ — mesmo contrato de turn

CLI/REPL, WebUI (SSE+WS com OIDC, rate limit, origin check), Slack, Telegram, webhooks — todos entrando pelos **mesmos contratos de turn** (`ProductAdapter`); a WebUI é **proibida de bypassar** as fronteiras de auth. A superfície como fronteira de segurança, não backdoor.

Hermes (rodada 2) — gateway multi-canal de processo único

TUI completa; **gateway multi-canal**: Telegram, Discord, Slack, WhatsApp, Signal, Email, iMessage, QQ, WeChat, Yuanbao — com continuidade cross-plataforma; **voz** (transcrição + TTS multi-provider); ACP para editores; servidor API OpenAI-compatível.

Goose (rodada 2) — desktop ACP sobre core embarcado

CLI completo + TUI; **desktop Electron falando ACP** com o core (binário embarcado, sem servidor separado); headless via recipes + scheduler; gateway Telegram e bot Discord; modo servidor MCP/ACP puro.

Aider (rodada 2) — input fora do terminal

CLI/REPL rica (prompt_toolkit, streaming markdown), browser UI (Streamlit), **watch mode** (`aider/watch.py`: comentários `ai!`/`ai?` no código de qualquer IDE viram comandos), **voz-para-código**, imagens/URLs no chat. A interface escapando para o editor de terceiros.

OpenHands (rodada 2) — control-plane SaaS

Web UI React (~40 rotas: conversas, settings, admin, billing, orgs); CLI `agent-canvas`; headless/REST via Agent Server; **resolvers GitHub/GitLab/Jira/Slack** (webhooks); enterprise/SaaS completo (Keycloak, Stripe, multi-tenant); deploy Docker/k8s.

n8n (rodada 2) — chat embarcável + canvas

Chat Trigger (app de chat hospedado + widget `@n8n/chat` embarcável + streaming), Manual Chat Trigger, webhooks arbitrários, editor visual (canvas) como interface de construção, MCP Server Trigger. O "harness invertido" cuja interface primária é o grafo.

Frameworks (rodada frameworks)

Os frameworks entregam o loop como biblioteca (a superfície programática pura) + streaming de eventos + human-in-the-loop como composição (OpenAI Agents SDK, LangGraph, CrewAI); a UI fica a cargo do integrador. É o extremo "só núcleo, superfície é sua" do espectro — o oposto do OpenClaw.