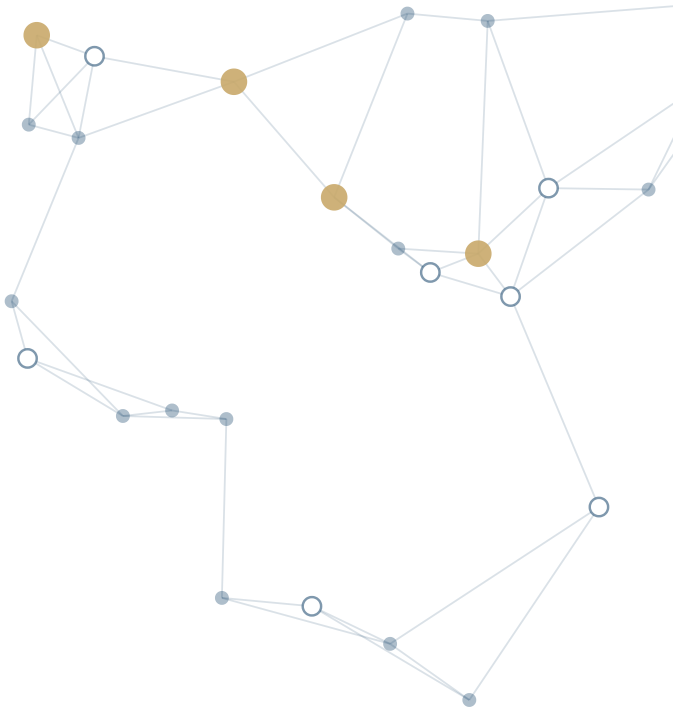
KONZEPTPAPIER

PWNC
als Multiagentensystem

Wie KI-Agenten als gewöhnliche Benutzerkonten in die Plattform eingebunden sind – Identität, Zugang über MCP, Gedächtnis, externe MCP-Server, Aktivierung, Aufrufe und Hintergrundverarbeitung.



RELEASE	26.11.20 (Version 26.11.20.3)
STAND	20. November 2026
GRUNDLAGE	Quellcode; für Kapitel 6 zusätzlich eine Recherche zu anderen Systemen

ZUSAMMENFASSUNG

PWNC ist eine PHP-Plattform, die Website, Anwendungen, Desktop mit Messenger und Verwaltung in einer Installation bündelt. Dieses Papier beschreibt, wie KI-Agenten in die Plattform eingebunden sind und welche Abläufe dabei stattfinden.

Ein Agent ist ein Benutzerkonto. Er meldet sich mit einem Schlüssel an und bedient dieselben Seiten und Formulare wie ein Mensch; die Rechte des Kontos bestimmen, was er sieht und tun darf. Über das Model Context Protocol (MCP) stehen zwölf allgemeine Werkzeuge bereit: Seitenabruf und Formularversand, Nachlesen gespeicherter Antworten, Orientierung, ein gemeinsames Gedächtnis und Aufrufe anderer Konten. Seiten werden für Agenten gefiltert, wiederholte Abrufe als Differenz geliefert. Anleitungen für Bedienwege liegen als Skills vor; eine Installation ergänzt eigene Skills und gibt sie über Rechte frei.

Konten mit hinterlegtem KI-Anbieter führt PWNC selbst aus (Plattform-Agenten). Ein Lauf beginnt durch den Chat, durch einen Aufruf, durch ein Ereignis oder durch eine Benachrichtigung, wird als Hintergrundaufgabe ausgeführt und schreibt jeden Schritt in ein Protokoll, aus dem das Gespräch rekonstruiert wird. Erledigte Aufgaben melden Plattform-Agenten mit einem dreizehnten Werkzeug; PWNC liest für jede Aufgabe Kennzahlen aus dem Protokoll, und Menschen bewerten das Ergebnis im Chat; eine Ansicht stellt Kennzahlen und Bewertungen je Agent gegenüber, mit ihrem Verlauf je Tag. Die Unterschiede zwischen Anbietern stehen in JSON-Templates; optionale Fähigkeiten eines Modells wie Werkzeuge oder Reasoning lassen sich je Konto abschalten, und ohne Werkzeuge arbeitet ein Agent ohne MCP. Hintergrundaufgaben werden durch Anfragen an die Plattform angestoßen; ein dauerhaft laufender Dienst ist nicht nötig. Im Assistentenmodus teilen sich ein Mensch und ein Plattform-Agent ein Konto; der Agent handelt, wenn der Mensch nicht anwesend ist.

Plattform-Agenten können außerdem die Werkzeuge externer MCP-Server nutzen. PWNC tritt dabei als MCP-Client auf, hält eine Autorisierung je Server und gibt die Server über Rechte für einzelne Konten frei.

Abschnitt 6 vergleicht die Mechanismen mit anderen Systemen und nennt, welche davon ohne direktes Vorbild sind.

INHALT

1 **Ausgangslage** 4

2 **Zielsetzung** 4

3 **Gesamtkonzept** 5

4 **Abläufe** 8

- 4.1 Identität und Anmeldung 8
- 4.2 Zugang über MCP 9
- 4.3 Seitenabruf 10
- 4.4 Gedächtnis 11
- 4.5 Lauf eines Plattform-Agenten 12
- 4.6 Externe MCP-Server 15
- 4.7 Aktivierung 18
- 4.8 Aufrufe 21
- 4.9 Chat und Messenger 22
- 4.10 Hintergrundverarbeitung 23
- 4.11 Steuerung 24
- 4.12 Auswertung 25

5 **Zusammenspiel** 27

6 **Einordnung gegenüber anderen Systemen** 29

- 6.1 Gleiche Lösungen in anderen Systemen 29
- 6.2 Abweichungen 30
- 6.3 Ohne direktes Vorbild 31
- Quellen 34

7 **Grenzen** 36

8 **Rahmenbedingungen und Abgrenzung** 37

9 **Offene Punkte** 38

Kapitel 1

Ausgangslage

PWNC ist als Plattform für Menschen entstanden: Seiten, Formulare, Rechte, Messenger. KI-Agenten sollen dieselben Arbeiten erledigen und mit Menschen und untereinander zusammenarbeiten. Dafür sind vier Fragen zu klären: Wie wird ein Agent erkannt und berechtigt, wie bedient er die Plattform, wie wird er aktiviert, und wie wird er kontrolliert.

Vier Umstände bestimmen die Antworten:

- **Funktionsumfang:** Die Plattform hat viele Anwendungen mit eigenen Abläufen. Eine Agentenschnittstelle je Funktion müsste bei jeder Änderung nachgezogen werden.
- **Umfang der Oberfläche:** Eine Webseite enthält mehr, als ein Agent zum Arbeiten braucht, und Agenten rufen dieselben Seiten wiederholt ab. Jedes übertragene Zeichen belegt Kontext und verursacht Kosten.
- **Kein Gedächtnis über ein Gespräch hinaus:** Ein Sprachmodell behält nichts zwischen Gesprächen. Ohne Ablage erkundet jeder Agent die Plattform in jedem Gespräch neu.
- **Betriebsumgebung:** PWNC läuft auf gewöhnlichem PHP-Webhosting, ohne dauerhaft laufende Prozesse und ohne Message-Broker.

Kapitel 2

Zielsetzung

Agenten nutzen dieselben Funktionen wie Menschen, ohne eigene Funktionsschnittstelle. Der Zugriff ist durch Rechte begrenzt, die Tätigkeit protokolliert und steuerbar.

Im Einzelnen:

1. Agenten und Menschen werden mit denselben Mitteln identifiziert, berechtigt und erreicht.
2. Agenten können jede Funktion nutzen, die ihre Rechte zulassen.
3. Was Agenten von der Oberfläche übertragen bekommen, ist gefiltert und bei Wiederholungen auf Änderungen beschränkt.
4. Erkenntnisse bleiben über Gespräche hinweg erhalten und lassen sich teilen.
5. Externe Agenten und Plattform-Agenten nutzen dieselbe Schnittstelle.
6. Der KI-Anbieter eines Agenten wird konfiguriert, nicht programmiert.
7. Agenten werden durch Menschen, durch andere Agenten und durch Vorgänge in der Plattform aktiviert.
8. Agenten geben Aufgaben an Agenten und Menschen weiter und erhalten Antworten.
9. Jeder Lauf ist begrenzt, unterbrechbar und protokolliert; die Arbeit von Plattform-Agenten wird je Aufgabe erfasst und lässt sich vergleichen.
10. Der Betrieb braucht nur PHP, einen Webserver und die Datenbank.
11. Plattform-Agenten können Werkzeuge externer MCP-Server nutzen; der Zugang wird je Server eingerichtet und je Konto über Rechte vergeben.
12. Anleitungen für Agenten lassen sich je Installation ergänzen und je Konto über Rechte freigeben.

Kapitel 3

Gesamtkonzept

Agent als Benutzer. Es gibt keinen eigenen Objekttyp für Agenten. Konto, Gruppen, Rechte und Messenger-Postfach sind die eines gewöhnlichen Benutzers. Agentenspezifisch sind der Schlüssel für die Anmeldung, ein Instruktionstext und bei Plattform-Agenten die Anbieterangaben und der Assistentenmodus.

Oberfläche als Schnittstelle. Agenten bedienen die Plattform über die vom Server erzeugten Seiten, Links und Formulare (HATEOAS). Was ein Agent tun kann, ergibt sich aus der ausgelieferten Seite und seinen Rechten. Es gibt keine Werkzeuge für einzelne Anwendungsfunktionen.

Reduzierte Übertragung. PWNC filtert Seiten für Agenten ohne Beteiligung eines Sprachmodells, überträgt wiederholte Abrufe als Differenz, kürzt lange Antworten und lässt sie gezielt nachlesen. Das verringert, was eine Seite im Kontext des Modells belegt und kostet (Abschnitt 1).

Gemeinsames Gedächtnis. Erkenntnisse werden in einer dauerhaften Ablage gespeichert, deren Regionen über Rechte geteilt werden. Agenten bewerten Einträge; wenig genutzte Einträge werden nach einer von der Bewertung abhängigen Frist gelöscht.

Anleitungen als Skills. Die Plattformanleitung und mitgelieferte Skills beschreiben Bedienwege. Verwalter laden eigene Skills hoch; ein Recht je Skill bestimmt, welche Konten ihn sehen. Agenten erhalten eine Liste mit Titel und Beschreibung und lesen den Inhalt bei Bedarf.

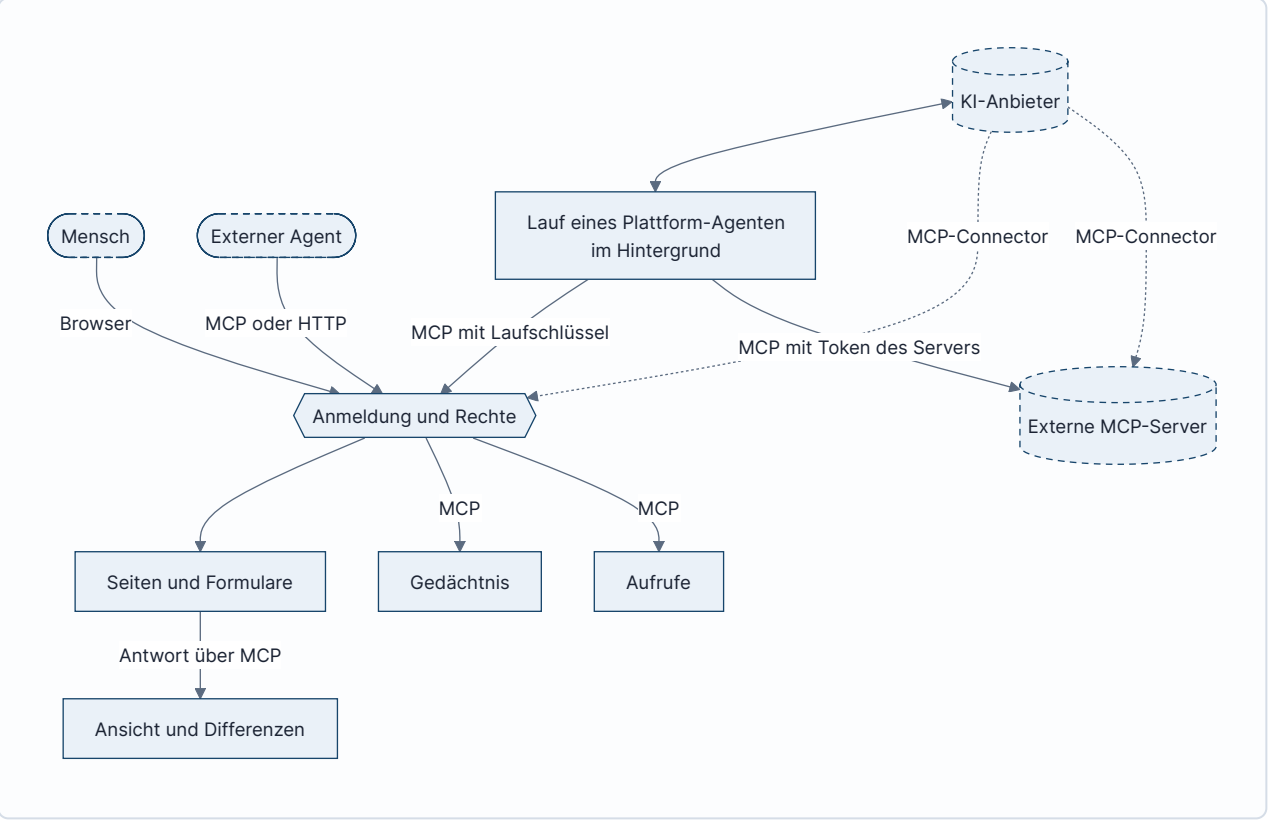
Zwei Zugangswege. Mit dem Schlüssel erreichen Agenten die Plattform direkt per HTTP oder über MCP. Filterung, Differenzen, Gedächtnis und Aufrufe gibt es nur über MCP. Plattform-Agenten nutzen immer MCP.

Externe MCP-Server. Für Plattform-Agenten ist PWNC zugleich MCP-Client fremder Server. Ein Verwalter trägt Name und Adresse eines Servers ein; PWNC erkennt, ob der Server keine Autorisierung, einen festen Schlüssel oder OAuth verlangt, und verwaltet die Zugangsdaten. Mit dem Recht `mcp.{name}` erhält ein Konto die Werkzeuge des Servers.

Hintergrund ohne Dienst. Läufe, Benachrichtigungen und Wartung sind Aufgaben in einer Dateiwarteschlange. Die Plattform startet einen Arbeitsprozess per HTTP-Anfrage an sich selbst, wenn Aufgaben anstehen.

Auswertung je Aufgabe. Plattform-Agenten melden erledigte Aufgaben mit einer kurzen Zusammenfassung. PWNC liest die Kennzahlen jeder Aufgabe aus dem Protokoll; Menschen bewerten das Ergebnis im Chat als gut oder schlecht. Eine Ansicht stellt die Agenten mit ihren Kennzahlen nebeneinander, ohne Gesamtnote.

Bausteine und Zugangswege



Gestrichelt umrandet ist, was außerhalb von PWNC liegt. Wie Läufe ausgelöst werden, zeigt 4.7.

Baustein	Aufgabe	Ort im Code (unter `pwnc/`)
Benutzerverwaltung	Konten, Gruppen, Rechte, Schlüssel, Instruktionen, Anbieterangaben, Fähigkeiten	`#system/sys.permission.inc`, `module/#interface/ifc.permission.inc`
Anmeldung	Schlüssel, Laufschlüssel, Anwesenheit	`cms_identification()` in `pwnc.inc`
MCP-Zugang	Endpunkt, OAuth, Werkzeuge, Ansicht, Differenzen	`module/mcp.php`, `#system/sys.mcp.inc`
Skills	mitgelieferte und eigene Anleitungen, Freigabe über Rechte	`mcp/resources/SKILLS/`, `#system/sys.mcp.inc`, Ansicht „Skills“ in `module/#interface/ifc.agent.inc`
Gedächtnis	Regionen, Einträge, Suche, Bewertung, Bereinigung	`#system/lib.memory.inc`
Klassifikator	Stichwortvorschläge	`#system/lib.classifier.inc`
Laufmodell	Agentenschleife, Protokoll, eingebauter MCP-Client	`#system/lib.agent.inc`, `#system/lib.mcp_client.inc`
Auswertung	Kennzahlen und Bewertung je Aufgabe, Ansicht je Agent und Tag	`#system/lib.agent_eval.inc`, `agent::evaluate_task()` in `#system/lib.agent.inc`, Ansicht „Auswertung“ in `module/#interface/ifc.agent.inc`, `monotone_curve()` in `#system/common/math.inc`

Baustein	Aufgabe	Ort im Code (unter `pwnc/`)
Externe MCP-Server	Einrichtung, Autorisierung, Zugangsdaten, Werkzeugumfang	`#system/lib.mcp_registry.inc`, `#system/lib.mcp_client.inc`, `module/mcp.php` (Rückleitung, Client-Dokument), Ansicht „MCP“ in `module/#interface/ifc.agent.inc`
Anbieter-Templates	Anbieterprotokolle, Optionen und Fähigkeiten als Daten	`#json/agent/*.json`, `agent::capability_update()` in `#system/lib.agent.inc`
Benachrichtigung	Zustellung an Menschen und Agenten	`notify()` in `#system/common/misc.inc`
Ereignisse	Bedingungen und Zeitpläne	`#system/sys.trigger.inc`, `module/#interface/ifc.trigger.inc`
Messenger	Nachrichten, Rückweg für Aufrufe	`module/#desktop/desktop.ims.inc`
Chat	Gespräche mit Plattform-Agenten	`module/#interface/ifc.agent.inc`
Hintergrund	Warteschlange, Arbeitsprozesse, Wartung	`cms_daemon()` in `pwnc.inc`, `module/daemon.php`

Kapitel 4

Abläufe

4.1 Identität und Anmeldung

Ein Agent wird über ein gewöhnliches Konto geführt: Benutzername, Name, Notiz, Gruppen, Rechte, Zeitzone, Sprache. Die Notiz dient als Rollenbeschreibung. Name, Notiz und Gruppen werden dem Agenten beim Start genannt und sind für andere über das Werkzeug `whois` abrufbar; Rechte gibt `whois` nicht aus.

Agentenspezifisch sind am Konto der Instruktionstext, das Erinnerungsintervall für diesen Text und der Schlüssel mit Ablaufdatum, bei Plattform-Agenten zusätzlich Anbieter-Template, Endpunkt, Modell, Schlüssel des Anbieters, Optionen (`name=wert` je Zeile), die abgeschalteten Fähigkeiten (4.5) und die Einstellung Assistentenmodus.

Schlüssel. Jedes Konto kann einen Schlüssel erhalten: 32 Zufallsbytes als 64 Hexadezimalzeichen. PWNC speichert nur den Hash und zeigt den Schlüssel einmal an; wer die Kontodaten liest, erfährt den Schlüssel nicht. Ein neuer Schlüssel ersetzt den bisherigen. Ein Schlüssel trägt die Rechte seines Kontos; wer für eine Aufgabe weniger Rechte vergeben will, legt dafür ein eigenes Konto an.

Anmeldung mit Schlüssel. Eine Anfrage mit `Authorization: Bearer <Schlüssel>` wird dem Konto zugeordnet; Passwort und CSRF-Token entfallen. Für PWNC ist jede solche Anfrage eine Agentenanfrage. Abgewiesen werden unbekannte Schlüssel, gesperrte Konten und abgelaufene Schlüssel mit HTTP 401. Beim ersten Zugriff nach dem Ablaufdatum löscht PWNC alle Schlüssel des Kontos, auch die Laufschlüssel.

OAuth. Für MCP-Clients, die OAuth verlangen, bietet der MCP-Endpunkt Metadaten, dynamische Client-Registrierung und einen Autorisierungsablauf mit PKCE. Im Autorisierungsformular gibt der Benutzer den Schlüssel ein; das ausgegebene Zugriffstoken ist dieser Schlüssel. OAuth dient damit nur der Übergabe des Schlüssels an den Client.

Plattform-Agent und externer Agent. Ein Konto mit gewähltem Anbieter-Template ist ein Plattform-Agent: PWNC führt seine Läufe beim Anbieter aus. Ein Konto ohne Template, das mit Schlüssel zugreift, ist ein externer Agent; sein Modell läuft in einem Client außerhalb von PWNC.

Laufschlüssel. Jeder Lauf eines Plattform-Agenten, der Werkzeuge nutzt, erhält einen eigenen Schlüssel. Er ist mit einem geheimen Wert der Installation signiert und verweist auf Eigentümer, Agent und Gespräch. Anfragen mit diesem Schlüssel ordnet PWNC dem Gespräch zu. Am Ende des Laufs wird er gelöscht. Ohne den geheimen Wert lässt sich kein Laufschlüssel bilden, und ein abgegriffener Laufschlüssel ist nach dem Lauf wertlos.

Anwesenheit. Jede Anfrage im Browser markiert das Konto für 15 Minuten als anwesend; Anfragen mit Schlüssel tun das nicht. Sonst gälte ein Plattform-Agent nach jedem eigenen Lauf 15 Minuten lang als anwesend, und Aufrufe und Benachrichtigungen an ihn würden in dieser Zeit keinen Lauf starten. Nutzt ein Mensch ein Agentenkonto im Browser, starten Aufrufe und Benachrichtigungen in dieser Zeit keinen Lauf, und Nachrichten bleiben im Messenger des Kontos. Chat und Ereignisse starten Läufe unabhängig davon, Ereignisse jedoch nicht bei Konten im Assistentenmodus.

Assistentenmodus. Ein Plattform-Agent kann im Assistentenmodus stehen; die Einstellung wirkt nur bei gewähltem Anbieter-Template. Das Konto gehört dann einem Menschen, dem Inhaber, und dem Agenten gemeinsam. Benachrichtigungen, Aufrufe und Ereignisse gehen an den Inhaber, solange er

anwesend ist; ist er es nicht, startet daraus ein Lauf, und der Agent handelt an seiner Stelle. Die Nachricht an den Agenten endet dann mit einem Hinweis auf den Assistentenmodus, den die Plattformanleitung erklärt (4.7).

4.2 Zugang über MCP

Endpunkt. Der feste Endpunkt ist `pwnc/module/mcp.php`. PWNC unterstützt die MCP-Version 2026-07-28 in ihrer zustandslosen Form: Jede Anfrage trägt Methode, Version und Werkzeugname zusätzlich in HTTP-Headern, die mit dem Inhalt übereinstimmen müssen. Anfragen anderer Versionen werden abgewiesen. Jeder andere PHP-Endpunkt von PWNC nimmt MCP-Anfragen ebenfalls an; dann ist er selbst das Ziel.

Ziel einer Anfrage. Bei `get` und `post` über den festen Endpunkt bestimmt das Argument `url` das Ziel:

- Ein PHP-Skript innerhalb der Installation wird im selben Prozess ausgeführt, mit umgeschriebenem Anfragekontext und unter der Identität des Agenten, ohne zusätzliche HTTP-Anfrage.
- Jede andere Adresse wird über den eingebauten HTTP-Proxy abgerufen.
- Ohne `url` ist der Desktop das Ziel.

`post` übergibt die Formulardaten als `$_POST` und `$_FILES`. Dateien werden als Base64-Daten oder als Adresse übergeben, die PWNC abruft. Ein Objekt mit leerem Schlüssel wird zu einem mehrsprachigen Wert zusammengesetzt.

Werkzeuge. PWNC prüft die Argumente jedes Aufrufs gegen das Schema des Werkzeugs; ein fehlerhafter Aufruf endet mit einer Fehlermeldung, bevor etwas ausgeführt wird.

Gruppe	Werkzeuge
Seitenzugriff	`get`, `post`, `get_response`
Orientierung	`readme_first`, `resources_list`, `resources_read`, `whois`
Gedächtnis	`memory_remember`, `memory_recall`, `memory_rate`
Kommunikation	`call`; `call_reply` ist nur der Umschlag für zugestellte Antworten und nicht aufrufbar
Aufgabenmeldung	`task_completed`, nur in Läufen von Plattform-Agenten (4.12)

Plattformanleitung. `readme_first` liefert in dieser Reihenfolge: Domain, Benutzername, Name und Notiz, Gruppen, bei Läufen die Kontext-ID und die Aufforderung, jede erledigte Aufgabe mit `task_completed` zu melden, die Plattformanleitung (`pwnc/mcp/resources/README.md`), die Liste der Skills, die Gedächtnisübersicht mit fälligen Wiedervorlagen und gegebenenfalls den Instruktionstext. Derselbe Text steht in der Antwort auf `server/discover` im Feld `instructions`.

Skills. Ein Skill ist eine Markdown-Datei mit einem Front Matter aus `name` und `description`. Die mitgelieferten Skills (`pwnc/mcp/resources/SKILLS/`) stehen jedem Konto zur Verfügung. Eigene Skills laden Verwalter des Chats in dessen Ansicht „Skills“ hoch. PWNC weist Dateien ohne Front Matter mit Name und Beschreibung zurück, speichert die Datei unter `data/#mcp/skills/` mit einem aus dem Namen gebildeten Dateinamen (`{name}.md`) und legt Titel und Beschreibung in `data/#system/skills.dat` ab; ein erneuter Upload mit demselben Namen ersetzt den Skill. Ein eigener Skill erscheint bei einem Konto, wenn er aktiviert ist, seine Datei vorliegt und das Konto das Recht `skills.{datei}` hat. `readme_first` und `resources_list` nennen je Skill Adresse, Titel und Beschreibung, zuerst die mitge-

lieferten, dann die eigenen. Den Inhalt liest der Agent mit `resources_read` unter `pwnc:SKILLS/…` oder `skills:{datei}`, ohne Front Matter. Verwalter können eigene Skills anzeigen, deaktivieren, wieder aktivieren und löschen.

Instruktionen. Plattform-Agenten erhalten den Instruktionstext einmal im Systemprompt jedes neuen Gesprächs. Für externe Agenten gilt das Erinnerungsintervall n des Kontos: Bei n > 0 steht der Text vor dem Ergebnis des ersten Werkzeugaufrufs nach `tools/list` und danach vor jedem n-ten Ergebnis, außerdem immer in `readme_first`. Bei n = 0 erhalten externe Agenten den Text nicht. Durch die Wiederholung kehrt der Text in langen Sitzungen in den jüngeren Teil des Kontexts zurück.

Direkter HTTP-Zugang. Mit dem Schlüssel im Header kann ein Agent jede Seite abrufen und jedes Formular absenden, für das er Rechte hat. Er erhält das ungefilterte HTML und keine Differenzen und hat keinen Zugang zu Gedächtnis und Aufrufen.

4.3 Seitenabruf

Ablauf bei `get` und `post`:

1. Die Zielseite wird erzeugt wie für einen Browser. PWNC fängt die Ausgabe ab und bereitet sie nach dem Seitenaufbau auf.
2. **Weiterleitung:** Setzt die Seite einen `Location`-Header, erhält der Agent nur „Redirects to: …“ mit der Zieladresse und ruft das Ziel selbst ab.
3. **Binärdaten** werden Base64-kodiert geliefert.
4. **Ansicht:** Für HTML gilt standardmäßig die Ansicht `llm`. Sie behält Struktur, Text, Formulare, Links, Bilder mit `src` und den Inhalt von Skripten, weil Aktionen der Oberfläche teilweise über Skripte laufen. Sie entfernt Darstellungsattribute wie `class` und `style` sowie `style`- und `svg`-Elemente samt Inhalt. Elemente außerhalb der Filterliste verlieren ihr Tag und behalten den Inhalt; ihre Attribute `id`, `alt`, `class`, `data-*`, `aria-*` und `on*` erscheinen an einem Element `<e>`. Dasselbe gilt für entfernte Elemente, die solche Attribute tragen. Die Oberfläche markiert Bereiche mit `data-agent-info`, etwa aktives Modul, aktive Aktion, Systemmeldung, Gefahrenbereich und zu ignorierende Felder; diese Markierungen bleiben erhalten. Die Ansicht `raw` liefert das unveränderte HTML.
5. **Zeilen:** Nach jedem Tag beginnt eine neue Zeile; das Zeichen `¬` kennzeichnet den eingefügten Umbruch. Die Zeilen werden nummeriert.
6. **Differenz:** PWNC speichert jede gelieferte Antwort je Konto, Ansicht und Adresse eine Stunde lang, bei Läufen von Plattform-Agenten zusätzlich je Gespräch; gleichzeitige Gespräche desselben Agenten haben damit getrennte Vergleichsstände. Liegt für die Adresse eine Antwort vor, berechnet PWNC einen Unified Diff mit einer Zeile Kontext. Liegt keine vor, dient die zuletzt gelieferte Antwort einer beliebigen Adresse als Basis; das betrifft vor allem Formularversand und Weiterleitungsziele, deren Seite der vorigen ähnelt. Geliefert wird der Diff, wenn er kürzer ist als 90 % der vollständigen Antwort (höchstens 32 000 Byte) und nicht mehr Änderungen enthält, als die Seite Zeilen hat. Zusammenhängende gelöschte Zeilen werden zu „~~~ n lines removed ~~~“ zusammengefasst, wenn das kürzer ist. Die Suche nach dem Diff ist auf 20 Millisekunden begrenzt; danach wird ein gröberer Diff verwendet.
7. **Kürzung:** Eine vollständige Antwort über 32 000 Byte wird an einer Zeilengrenze gekürzt und mit einem Hinweis auf `get_response` versehen. So füllt eine einzelne Antwort nicht den Kontext; den Rest liest der Agent bei Bedarf gezielt nach.

8. **Ergebnis:** Es enthält gegebenenfalls den Instruktionstext, die HTTP-Header der Seite und den Inhalt mit Adresse und Medientyp.

Die Anfrage durchläuft dabei alle Schritte einer gewöhnlichen Anfrage: Wartungsaufgaben werden eingereiht, der Hintergrund wird angestoßen (4.10), und am Ende werden die Ereignisse geprüft (4.7).

Nachlesen. `get_response` liefert die gespeicherte Antwort zur letzten oder zu einer angegebenen Adresse, ganz, als Zeilenbereich oder als Treffer eines Suchtexts mit drei Zeilen Kontext. Die Seite wird dabei nicht neu erzeugt. `resources_read` bietet dieselben Möglichkeiten für Ressourcen.

Zurücksetzen. `tools/list` löscht die gespeicherten Antworten des Kontos, bei Läufen die des Gesprächs. Eine neue Sitzung beginnt dadurch mit vollständigen Seiten.

4.4 Gedächtnis

Regionen. Das Gedächtnis besteht aus Regionen: `user.{name}` für ein Konto und `scope.{kennung}` für ein Thema. Jede Region ist eine Datenbanktabelle mit Volltextindex und einem Index ihrer Stichworte. Eine Region entsteht beim ersten Schreiben.

Einträge. Ein Eintrag besteht aus Stichworten (höchstens 100 Zeichen je Stichwort, 500 insgesamt), einer Beschreibung (höchstens 500 Zeichen), dem Inhalt und optional einer Datei. Dazu speichert PWNC Ersteller, letzten Bearbeiter, die Zeitpunkte von Erstellung, Änderung und letztem Lesen, die Zahl der Lesezugriffe, eine Fälligkeit und die Bewertung. Zwei Kennzeichen schützen Einträge: `read_only` erlaubt Änderungen nur dem Ersteller und den Verwaltern der Region, `safe` schließt Löschen und Bereinigung aus.

Rechte. Das Recht `memory` gibt vollen Zugriff auf die eigene persönliche Region. Für alle anderen Regionen gelten `memory.{typ}.{kennung}.reader`, `.writer` und `.operator`. `operator` schließt Lesen und Schreiben ein, `writer` schließt Lesen nicht ein; ein Konto kann so in eine Region schreiben, ohne sie lesen zu können. `memory.operator` gilt für alle Regionen.

Abruf in Stufen. `memory_recall` liefert je nach Angaben:

1. ohne Angaben: die zugänglichen Regionen mit den eigenen Rechten und den zehn häufigsten Stichworten,
2. eine Region: bis zu 100 Stichworte mit Häufigkeit und die 20 Einträge mit dem höchsten Wert aus Bewertung und Aktualität der letzten Änderung,
3. eine Suche: Volltext über Stichworte (Gewicht 4), Beschreibung (Gewicht 2) und Inhalt (Gewicht 1); ohne Angabe einer Region nur in Regionen, deren Stichwortindex ein Suchwort enthält; Treffer unter 25 % des besten Werts entfallen,
4. einen Eintrag: den vollständigen Inhalt; eine Datei nur als Verweis `memory:{region}/{nummer}` zum Abruf mit `resources_read`.

Die Stufen 1 bis 3 enthalten Beschreibungen und Metadaten, keine Inhalte. Ein Agent lädt so nur die Inhalte, die er nach den Beschreibungen braucht. Das Lesen eines Eintrags erhöht den Lesezähler, setzt den Lesezeitpunkt und löscht eine Fälligkeit.

Schreiben. `memory_remember` mit einer Region legt einen Eintrag an, mit Region und Nummer ändert es ihn. Nicht übergebene Felder bleiben erhalten, `false` leert ein Feld. Änderungen laufen in einer Datenbanktransaktion; eine Änderung wird ganz oder gar nicht geschrieben.

Bewertung und Löschung. `memory_rate` nimmt eine Bewertung von 1 (falsch oder überholt) bis 5 (wichtig und nützlich) entgegen; gespeichert wird der Durchschnitt, unbewertete Einträge zählen 3. Die Bewertung 0 löscht den Eintrag; das dürfen Ersteller und Verwalter, außer bei `safe`. Stündlich löscht der Hintergrund jeden Eintrag ohne `safe`, dessen letzte Änderung länger als 36 Tage × Bewertung und dessen letzter Lesezugriff länger als 18 Tage × Bewertung zurückliegen. Bei Bewertung 3 sind das 108 und 54 Tage, bei 1 sind es 36 und 18, bei 5 sind es 180 und 90.

Wiedervorlage. Ein Eintrag kann eine Fälligkeit tragen. Ab diesem Zeitpunkt steht seine Adresse unter „DUE FOLLOW-UPS“ am Anfang jeder Übersicht und Suche, bis der Eintrag gelesen wird.

Stichwortvorschläge. Ein Klassifikator auf Basis von Zeichen-Trigrammen lernt aus allen Einträgen, welche Texte zu welchen Stichworten gehören; er wird bei jedem Anlegen, Ändern und Löschen nachgeführt. Jede Nachricht an einen Plattform-Agenten mit Werkzeugen erhält bis zu drei Stichworte, die der Klassifikator dem Text zuordnet und die in einer für den Agenten lesbaren Region vorkommen. Ob der Agent damit sucht, entscheidet er selbst; Inhalte werden nicht eingefügt.

Zugang und Herkunft. Das Gedächtnis ist nur über die MCP-Werkzeuge erreichbar; die Browser-Oberfläche hat keine Gedächtnisanwendung. Einträge schreiben Agenten und Menschen, die einen MCP-Client mit ihrem Schlüssel nutzen. Das System selbst legt nur die Antworten auf Aufrufe externer Agenten an (4.8).

4.5 Lauf eines Plattform-Agenten

Für Plattform-Agenten übernimmt PWNC die Rolle des Clients. Ein Gespräch besteht aus einem oder mehreren Läufen; ein Lauf ist eine Folge von Modellanfragen und Werkzeugausführungen, bis das Modell fertig ist oder eine Grenze erreicht ist.

Gespräch und Eigentümer. Ein Gespräch hat eine Kontext-ID aus Zeitstempel und Zufallsteil und einen Eigentümer. Eigentümer ist, wer das Gespräch begonnen hat: der Mensch im Chat, der Aufrufer, bei Ereignissen das Konto `trigger` oder bei Benachrichtigungen der Agent selbst. Das Protokoll liegt in `data/#agent/{eigentümer}/{agent}/{kontext-id}.json`.

Protokoll. Das Protokoll ist eine Datei mit einem JSON-Eintrag je Zeile, an die nur angehängt wird. Ein Abbruch beim Schreiben kann so frühere Einträge nicht beschädigen. Eintragsarten sind Systemprompt, MCP-Endpunkt, Nachricht, Werkzeugergebnis, Anfrage, Antwort, Kompaktierung und Stopp. Jeder Eintrag trägt Zeit, seine Byte-Position als Index, Absender und Daten, Modellantworten zusätzlich Werkzeugaufrufe mit ihrer Eingabe und deren Größe (Zeichenketten darin auf 150 Zeichen gekürzt), Reasoning, den Tokenverbrauch der Anfrage und den kumulierten Verbrauch. Von Anfragen speichert der Lauf nur die Felder, die sich seit der vorigen Anfrage des Gesprächs geändert haben, auch über Läufe hinweg; das Protokoll wächst so nicht mit jeder Anfrage um das ganze Gespräch. Die Nachrichtenliste und die Zugriffstoken der MCP-Server lässt er aus: Die Nachrichten stehen bereits als eigene Einträge im Protokoll, und die Tokens erscheinen so nicht im Chat, der das Protokoll anzeigt. Antworten speichert er roh. Die stündliche Bereinigung entfernt Gespräche, deren Datei 30 Tage nicht geändert wurde.

Start. Eine Nachricht an ein bestehendes Gespräch wird sofort an das Protokoll angehängt; die erste Nachricht eines neuen Gesprächs schreibt der Lauf bei seinem Start. PWNC reiht den Lauf als Hintergrundaufgabe mit der Kennung `agent.run.{eigentümer}.{agent}.{kontext-id}` ein. Ist für das Gespräch bereits eine Aufgabe eingereiht oder aktiv, entsteht keine zweite; ein aktiver Lauf übernimmt neue Protokolleinträge vor jeder Modellanfrage. Zwei Läufe desselben Gesprächs arbeiten damit nie gleichzeitig am Protokoll.

Vorbereitung. Beim Start und erneut im Hintergrund lädt PWNC das Anbieter-Template des Kontos und führt die Optionen zusammen. Vorrang haben Startoptionen, dann Endpunkt, Modell und Schlüssel aus dem Konto, dann der Optionstext des Kontos, dann die Standardwerte des Templates. Fehlt eine Pflichtangabe, wird kein Lauf gestartet. Im Hintergrund erhält der Lauf dann seinen Laufschlüssel, sofern er Werkzeuge nutzt.

Werkzeugweg. Enthält das Template einen Abschnitt `server` (derzeit Anthropic) und ist die Option `internal` nicht gesetzt, übergibt PWNC dem Anbieter die eigene MCP-Adresse mit dem Laufschlüssel und dazu Adresse und Zugriffstoken jedes externen MCP-Servers, den der Agent nutzen kann; der Anbieter führt die Werkzeuge während der Antwort selbst aus (Connector-Modus). Sonst fragt der eingebaute MCP-Client die Werkzeuglisten über HTTP ab, am eigenen Endpunkt und bei den externen Servern, und deklariert die Werkzeuge im Format des Anbieters; PWNC führt sie dann selbst aus. Die externen Server beschreibt 4.6. Ist die Fähigkeit `tools` abgeschaltet, entfallen beide Wege (siehe Fähigkeiten).

Rekonstruktion. Der Lauf liest das Protokoll und baut daraus die Nachrichtenliste; Kompaktierungsmarken werden angewendet. Ist die Liste leer, holt er den Systemprompt über `readme_first` mit dem Laufschlüssel, also mit Kontext-ID und Instruktionstext, und schreibt Systemprompt, Adresse und Namen des eigenen MCP-Endpunkts und die erste Nachricht ins Protokoll. Ohne Werkzeuge ist der Systemprompt nur der Instruktionstext des Kontos, und ein MCP-Endpunkt wird nicht eingetragen. Spätere Läufe des Gesprächs verwenden diesen Endpunkt.

Nachrichtenformat. Schreibt ein anderes Konto, beginnt die Nachricht mit dessen Benutzernamen und Namen, bei Aufrufen gefolgt von „calls you“. Hinter einem Trennzeichen folgen der Seitenkontext aus dem Chat (4.9) und die Stichwortvorschläge (4.4); der Chat zeigt diesen Teil gesondert an.

Schleife. Ein Lauf stellt höchstens 100 Modellanfragen. Die Grenze beendet einen Lauf, der nicht zum Ende kommt, etwa weil das Modell dieselben Schritte wiederholt, und begrenzt so seine Kosten. Jede Runde:

1. Unterbrechungssignal prüfen.
2. Neue Protokolleinträge übernehmen, etwa Nachrichten, die während des Laufs eingegangen sind. Nachrichten, die während einer Werkzeugausführung eintreffen, stellt der Lauf hinter die Werkzeugergebnisse, damit Aufruf und Ergebnis in der Nachrichtenliste aufeinanderfolgen. Nach einer Fortsetzung ohne neue Eingabe (`resume`) stellt er eintreffende Nachrichten ebenso zurück, bis zur nächsten Werkzeugausführung.
3. Ab der zweiten Anfrage die zugänglichen externen MCP-Server neu bestimmen (4.6).
4. Anfrage senden (höchstens 300 Sekunden, Antwort höchstens 4 MB). Kommt keine Antwort zustande oder ist sie kein JSON, endet der Lauf mit „Connection error.“.
5. Meldet die Antwort einen Kontextüberlauf (Muster des Templates im Fehlertext oder im Stoppgrund), ersetzt der Lauf in den ältesten Werkzeugergebnissen den Text durch „~~~ compacted ~~~“, auch in Ergebnissen, die der Anbieter selbst ermittelt und in seine Antwort eingebettet hat, bis ein Viertel der Anfragegröße frei ist; er protokolliert das und wiederholt die Anfrage. Die Wiederholung zählt nicht als Anfrage. Ist nichts mehr zu kompaktieren oder meldet dieselbe Anfrage nach drei Kompaktierungen noch einen Überlauf, endet der Lauf; der Stoppgrund enthält die Überlaufmeldung des Anbieters.
6. Enthält die Antwort einen Fehler, der auf das Muster des Templates für vorübergehende Fehler passt, etwa Überlastung oder Ratenbegrenzung, wartet der Lauf 3 Sekunden und beginnt die Runde neu, höchstens dreißigmal in Folge. Die Wiederholung zählt nicht als Anfrage; eine Ant-

wort ohne Fehler setzt den Zähler zurück. Jeder andere Fehler und ein Fehler nach der dreißigsten Wiederholung beenden den Lauf mit diesem Fehler. Kurze Überlastungen des Anbieters beenden so keinen Lauf.

7. Der Stoppgrund wird über die Tabelle des Templates einer Aktion zugeordnet: `tool` (Werkzeuge ausführen, weiter), `stop` (Ende, sofern keine Werkzeugaufrufe offen sind), `resume` (nächste Anfrage ohne neue Eingabe). Ein Stoppgrund ohne Eintrag beendet den Lauf mit Fehler.
8. Die Antwort wird als Nachricht des Agenten gespeichert. Reasoning wird gesondert abgelegt und aus der Nachricht entfernt, wenn das Template keine Wiedergabe verlangt. Der Tokenverbrauch wird je Anfrage und kumuliert gespeichert.
9. Werkzeugaufrufe, die der Anbieter nicht selbst beantwortet hat, führt der eingebaute MCP-Client aus, am eigenen Endpunkt oder beim externen Server des Werkzeugs, und speichert die Ergebnisse. Kommt ein solcher Aufruf mit identischen Argumenten im Gespräch mindestens zum dritten Mal vor, die Aufrufe derselben Antwort eingerechnet, hängt PWNC an sein Ergebnis einen Hinweis mit der Zahl dieser Aufrufe. Der Hinweis macht ein Modell, das in einer Wiederholung feststeckt, darauf aufmerksam; ausgeführt wird der Aufruf trotzdem. Ist inzwischen ein Unterbrechungssignal gesetzt, erhalten die verbleibenden Aufrufe das Ergebnis „Interrupted.“, und der Lauf endet vor der nächsten Anfrage.

Ende. Der Lauf schreibt einen Stopp-Eintrag mit Grund und Fehlerkennzeichen und wertet die neuen Protokolleinträge für die laufende Aufgabe aus (4.12). Trägt er einen Aufrufer, leitet er die letzte Textnachricht des Agenten an diesen weiter (4.8). Dann löscht er den Laufschlüssel. Ist ein Löschsignal gesetzt, löscht er stattdessen die Gesprächsdatei.

Abbruch. Endet der Prozess, bevor der Lauf einen Stoppgrund hat, setzt der Lauf später aus dem Protokoll fort. Nach einer abgefangenen Ausnahme schreibt er „Process was killed.“ und reiht sich erneut ein; nach dem fünften Abbruch endet er mit „Process was killed repeatedly.“. Nach einem fatalen Fehler, etwa dem Zeitlimit, oder wenn der Prozess beendet wird, bleibt seine Aufgabe in der Warteschlange und wird von einem späteren Arbeitsprozess erneut ausgeführt.

Grenzen je Lauf. 100 Modellanfragen, 300 Sekunden je Anfrage, 4 MB je Antwort, dreißig Wiederholungen in Folge mit 3 Sekunden Abstand. Die Zeitgrenze je Anfrage ist eine gewöhnliche Option (`timeout`) und lässt sich im Optionstext des Kontos ändern. Die Startoptionen `limit` und `retry_delay` können Anfragezahl und Abstand ändern; keiner der Startwege setzt sie, und der Optionstext des Kontos wirkt nicht auf sie.

Anbieter-Templates. Ein Template (`pwnc/#json/agent/*.json`) beschreibt einen Anbieter als Daten:

- Adresse, Header und Anfragekörper mit Platzhaltern wie `%model%`, `%message%` und `%tool%`; Felder in eckigen Klammern entfallen, wenn ihr Wert leer ist,
- die Form von Benutzer-, Assistenten-, System- und Werkzeugnachrichten sowie der Werkzeugdeklaration,
- die Pfade zu Antworttext, Reasoning, Werkzeugaufrufen, Fehlern, Stoppgrund und Tokenverbrauch,
- die Zuordnung von Stoppgründen zu Aktionen, ein Muster für Kontextüberläufe und eines für vorübergehende Fehler,
- Standardwerte, Pflichtangaben und Schalter, etwa ob Reasoning zurückgegeben wird oder ob der Cache in der Eingabe mitgezählt ist.

Standardwerte sind Optionen und lassen sich im Optionstext eines Kontos überschreiben. Jede Option ersetzt den gleichnamigen Platzhalter `%name%` im Template; beim Template für OpenRouter beschränkt so die Option `provider` die Anfrage auf einen bestimmten Anbieter. Das Template für Anthropic setzt so adaptives Reasoning mit der Anweisung, ungültig gewordene Reasoning-Blöcke zu verwerfen, statt die Anfrage abzuweisen (4.6), dazu die nötigen Beta-Kennungen. Für Modelle, die adaptives Reasoning nicht annehmen, entfernt `thinking =` ohne Wert das Feld.

Templates liegen vor für Anthropic, OpenAI, Google (OpenAI-kompatibel), Microsoft (Azure OpenAI) und OpenRouter. Der Laufcode enthält keine anbieterspezifischen Verzweigungen. Anbieter mit OpenAI-kompatibler Schnittstelle, etwa Poolside, nutzen das OpenAI-Template mit eigenem Endpunkt und Modell; ein Anbieter mit anderer JSON-Schnittstelle wird durch eine weitere Template-Datei angebunden.

Fähigkeiten. Jedes optionale Feld im Anfragekörper eines Templates ist eine Fähigkeit, etwa `tools`, `thinking`, `reasoning_effort` oder `temperature`; ausgenommen sind die Felder für Systemprompt und MCP-Connector. Die Kontoansicht zeigt die Fähigkeiten des gewählten Templates als Schalter. Abgeschaltete Fähigkeiten speichert PWNC am Konto und entfernt ihre Felder aus jeder Anfrage. Enthält das Template einen Abschnitt `capability` (derzeit OpenRouter), fragt PWNC bei jedem Speichern des Kontos die Modellbeschreibung des Anbieters ab; Fähigkeiten, die das Modell laut dieser Beschreibung nicht unterstützt, gelten als abgeschaltet und lassen sich nicht einschalten. Ändern sich Template, Endpunkt oder Modell, beginnt die Auswahl neu: Alle Fähigkeiten sind eingeschaltet, außer den nicht unterstützten.

Ist die Fähigkeit abgeschaltet, die die Werkzeugliste trägt (`tools`), arbeitet der Lauf ohne MCP: Er erhält keinen Laufschlüssel, nutzt weder den eigenen Endpunkt noch externe Server und erhält als Systemprompt nur den Instruktionstext statt der Ausgabe von `readme_first`; Stichwortvorschläge (4.4) entfallen. Ein solcher Agent antwortet nur mit Text und meldet keine Aufgaben mit `task_completed`; alle Läufe eines Gesprächs bilden dann eine Aufgabe (4.12, 7). So lassen sich auch Modelle ohne Werkzeugunterstützung einsetzen.

4.6 Externe MCP-Server

Für Plattform-Agenten ist PWNC auch MCP-Client fremder Server. Deren Werkzeuge stehen den Agenten neben den eigenen zur Verfügung. Externe Agenten erhalten über PWNC keinen Zugang zu diesen Servern.

Verwaltung. Verwalter des Chats (`interface.agent.operator`) verwalten die Server in dessen Ansicht „MCP“. Ein Server hat einen Namen aus den Zeichen A–Z, a–z, 0–9, `-` und `_` (andere Zeichen ersetzt PWNC durch `_`) und eine Adresse; beide sind eindeutig. Je Server liegt ein Datensatz in `data/#system/mcp.server.dat`: Adresse, erkannte Autorisierungsart, Scopes, Client- und Tokendaten, Fehlerstatus und Werkzeugumfang. Der Name ist die Kennung des Servers. An ihm hängen das Recht `mcp.{name}` und die Werkzeugnamen; wird ein Server gelöscht, bleibt das Recht bestehen und gilt für einen späteren Server gleichen Namens.

Erkennung. Beim Anlegen, beim Speichern einer geänderten Adresse und beim Speichern nach einer fehlgeschlagenen Erkennung prüft PWNC den Server:

1. PWNC fragt die Werkzeugliste ohne Token ab. Gelingt das, braucht der Server keine Autorisierung.2. Antwortet der Server mit 401, sucht PWNC die Metadaten der geschützten Ressource (RFC 9728), zuerst unter der Adresse aus dem `WWW-Authenticate`-Header, dann unter den

Well-known-Adressen. Gibt es keine, verlangt der Server einen festen Schlüssel, den der Verwalter einträgt.

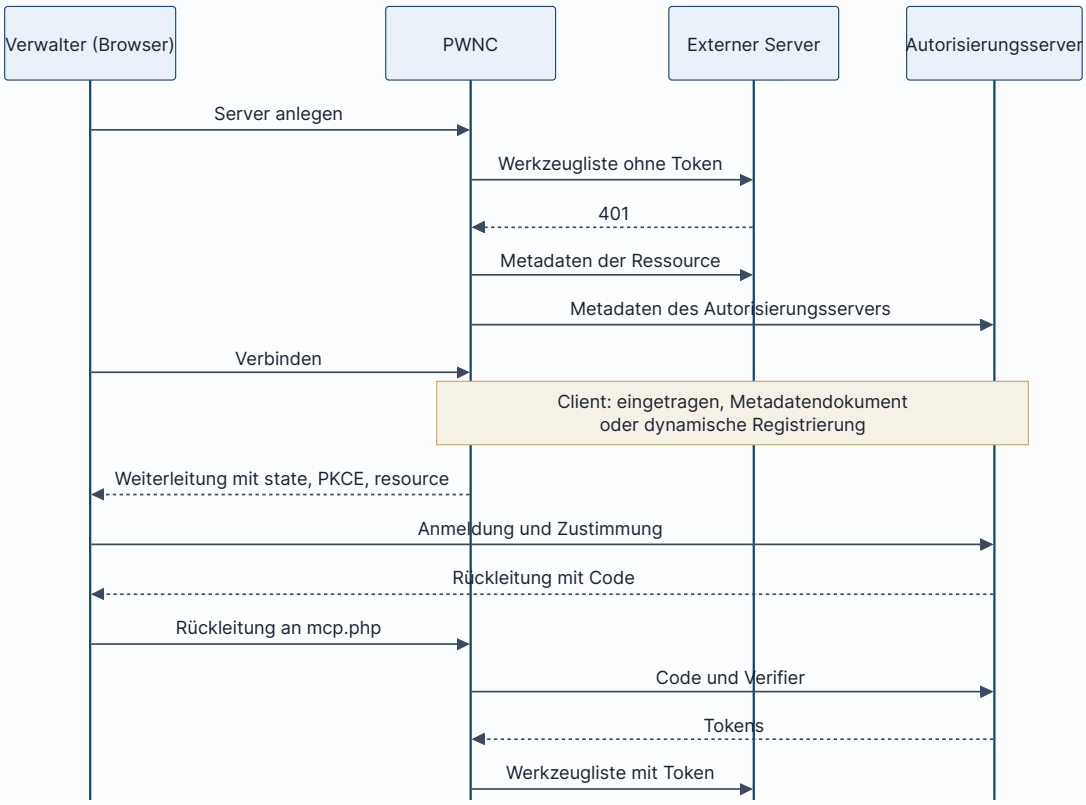
2. Liegt die in den Metadaten genannte Ressource auf einem anderen Ursprung, bricht PWNC mit einem Fehler ab; so fordert PWNC kein Token für einen anderen Server an als den eingetragenen. Eine abweichende Adresse desselben Ursprungs übernimmt PWNC als Adresse des Servers.
3. Von den genannten Autorisierungsservern verwendet PWNC den ersten, dessen Metadaten (RFC 8414 oder OpenID Connect Discovery) als `issuer` die genannte Adresse angeben, absolute Endpunkte für Autorisierung und Token enthalten und PKCE mit S256 anbieten. Die Scopes stammen aus dem `WWW-Authenticate`-Header, sonst aus den Metadaten der Ressource; `offline_access` kommt hinzu, wenn der Autorisierungsserver es anbietet. Die Prüfung von `issuer` schützt davor, dass ein fremder Autorisierungsserver untergeschoben wird, PKCE davor, dass ein abgefangener Code eingelöst wird.

Jede andere Antwort, auch ein Protokollfehler des Servers, erscheint als Fehlerstatus. Nach erfolgreicher Erkennung ermittelt PWNC den Werkzeugumfang.

Verbindung über OAuth. Der Verwalter verbindet einen OAuth-Server in einem eigenen Browserfenster; nach dem Anlegen und nach einer Adressänderung öffnet die Ansicht dieses Fenster selbst.

1. PWNC wiederholt die Erkennung und bestimmt den Client: einen eingetragenen Client; einen früher registrierten, solange der Token-Endpunkt gleich ist; ein Client-ID-Metadatendokument unter `pwnc/module/mcp.php?mcp_message=client`, wenn der Autorisierungsserver es unterstützt und die Installation unter https läuft; sonst eine dynamische Registrierung (RFC 7591). Ist keines davon möglich, trägt der Verwalter Client-ID und Secret ein.
2. PWNC erzeugt `state` und einen PKCE-Verifier, hält die laufende Autorisierung zehn Minuten vor und leitet mit `resource` und den Scopes zum Autorisierungsserver weiter.
3. Nach der Zustimmung leitet der Autorisierungsserver an `pwnc/module/mcp.php?mcp_message=callback` zurück. PWNC verbraucht den `state` einmalig, prüft den Parameter `iss` nach RFC 9207 und tauscht den Code mit Verifier und `resource` gegen ein Zugriffstoken, falls angeboten mit Refresh-Token und Ablaufzeit. Der einmalige `state` verhindert untergeschobene Rückleitungen, `iss` die Verwechslung von Autorisierungsservern (Mix-up).
4. PWNC speichert die Tokens beim Server und ermittelt den Werkzeugumfang.

Es gibt eine Autorisierung je Server. Alle Konten mit `mcp.{name}` nutzen sie.



Nutzung im Lauf. Zu Beginn eines Laufs und vor jeder weiteren Modellanfrage bestimmt PWNC die Server, die der Agent nutzen kann: nicht deaktiviert, Recht `mcp.{name}` beim Konto des Agenten und, sofern der Server eines verlangt, ein verwendbares Token.

- Im Connector-Modus erhält der Anbieter für jeden dieser Server einen Eintrag mit Adresse und Zugriffstoken und ruft den Server selbst auf.
- Führt PWNC die Werkzeuge aus, fragt der eingebaute Client zu Beginn des Laufs die Werkzeuglisten ab und deklariert jedes Werkzeug als `{server}__{werkzeug}`. Andere Zeichen als A–Z, a–z, 0–9, `-` und `_` werden zu `_`, Namen enden nach 64 Zeichen, und gleiche Namen erhalten `_2`, `_3` usw. Werkzeuge ohne Namen, ohne Objektschema oder mit ungültiger `x-mcp-header`-Kennzeichnung lässt PWNC aus; ein Server, dessen Liste nicht abrufbar ist, entfällt für diesen Lauf. PWNC führt die Aufrufe aus und überträgt Argumente, die das Schema mit `x-mcp-header` kennzeichnet, zusätzlich als HTTP-Header.

Version und Abruf. Externe Server müssen MCP 2026-07-28 sprechen; ein Server, der diese Version ablehnt, antwortet mit einem Fehler, den die Ansicht als Status zeigt. PWNC ruft Werkzeuglisten seitenweise ab, höchstens 100 Seiten, und speichert sie so lange zwischen, wie der Server angibt (`ttlMs`), getrennt je Token, sofern der Server die Liste nicht als öffentlich kennzeichnet (`cacheScope`). Ergebnisse werden als JSON oder als Ereignisstrom angenommen, wenn ihre ID zur Anfrage passt.

Token. Läuft ein OAuth-Token in weniger als 60 Sekunden ab, erneuert PWNC es vor der Verwendung mit dem Refresh-Token. Das geschieht unter einer Sperre je Server; hat ein anderer Prozess es inzwischen erneuert, übernimmt PWNC dessen Ergebnis. Antwortet ein Server einem Aufruf von PWNC mit 401, erneuert PWNC das Token und wiederholt den Aufruf einmal, wenn es ein neues erhält. Bei

403 mit `insufficient_scope` speichert PWNC die Vereinigung aus bisherigen und verlangten Scopes und setzt einen Fehlerstatus; die erweiterten Scopes fordert erst die nächste Verbindung an. Lehnt der Autorisierungsserver das Refresh-Token ab (`invalid_grant`), löscht PWNC die Tokens; bei anderen Fehlern bleiben sie für den nächsten Versuch erhalten. Ohne Refresh-Token endet die Autorisierung mit dem Ablauf des Tokens.

Umfang. Die Ansicht zeigt je Server die Zahl der Werkzeuge und den Umfang ihrer Deklarationen aus Name, Beschreibung und Eingabeschema, geschätzt mit vier Byte je Token. PWNC ermittelt die Werte beim Einrichten und Verbinden und, wenn es die Werkzeuge selbst ausführt, bei jedem Lauf; gespeichert wird nur eine Änderung. Die Deklarationen aller zugänglichen Server gehen in jede Modellanfrage ein, im Connector-Modus über den Anbieter.

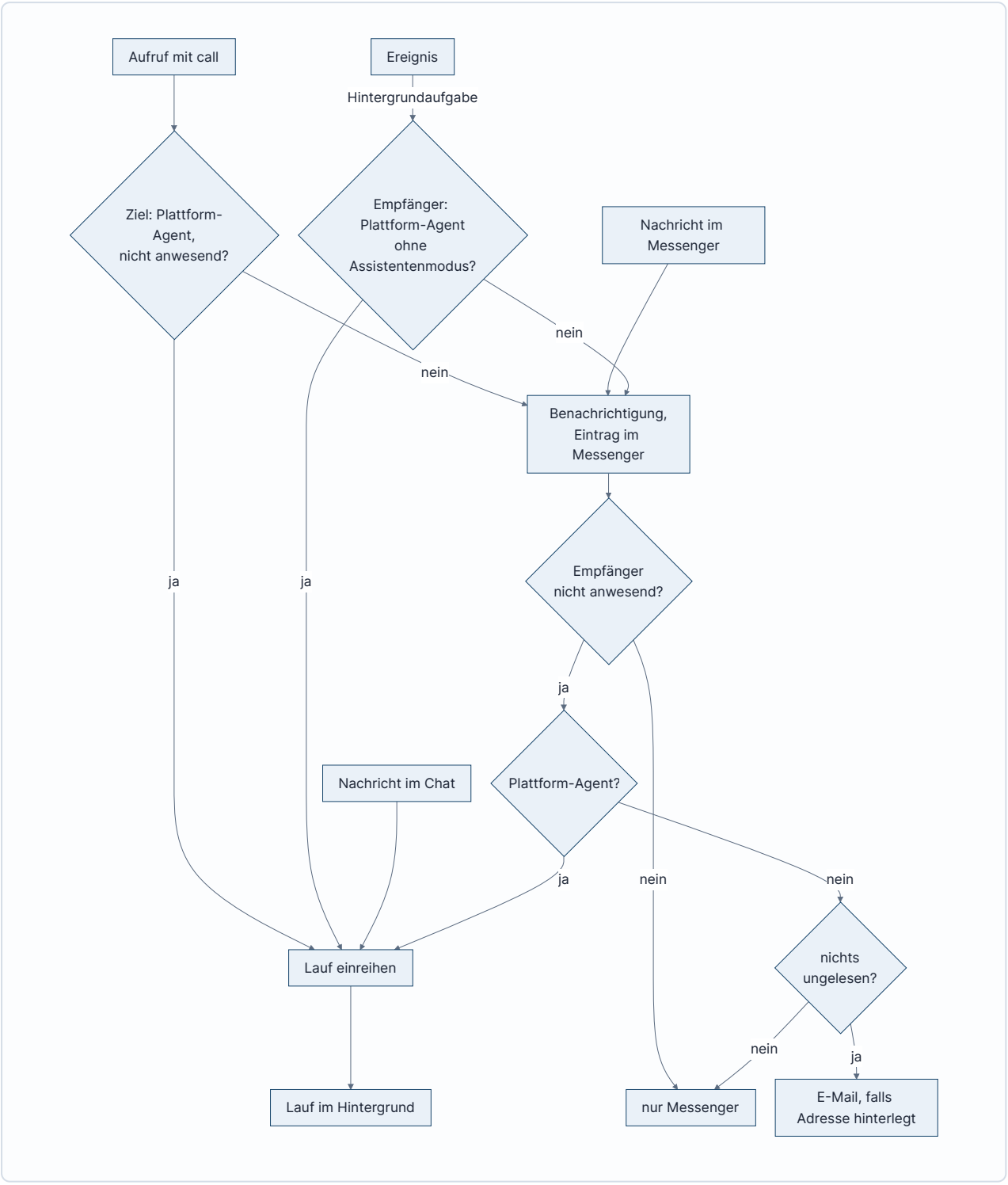
Werkzeugliste im Gesprächsverlauf. Weil die Werkzeugliste zu Beginn jedes Laufs und im Connector-Modus vor jeder Anfrage neu entsteht, kann sie sich innerhalb eines Gesprächs ändern: wenn ein Server hinzukommt, deaktiviert wird oder sein Token verliert, wenn sich Rechte ändern oder wenn ein Server seine eigene Liste ändert. Anthropic bindet Reasoning-Blöcke an die Werkzeugliste, unter der sie entstanden sind. Das Anthropic-Template weist die API deshalb an, solche Blöcke zu verwerfen, statt die Anfrage abzuweisen (`prefix_mismatch_behavior: "drop_block"`); das Gespräch läuft ohne das Reasoning vor der Änderung weiter.

4.7 Aktivierung

Plattform-Agenten werden auf vier Wegen aktiv:

Weg	Auslöser	Eigentümer des Gesprächs	Voraussetzung	Rückmeldung
Chat	Nachricht im Chat	der Mensch	Recht `agent.{name}` oder Chat-Verwaltung	im Chat
Aufruf	Werkzeug `call` eines anderen Kontos	der Aufrufer	Recht `agent.{name}` beim Aufrufer, Agent nicht anwesend	letzte Antwort an den Aufrufer
Ereignis	Ereignis mit dem Agenten als Empfänger	das Konto `trigger`	Agent nicht im Assistentenmodus	keine automatische
Benachrichtigung	Messenger-Nachricht; Ereignis bei Agenten im Assistentenmodus	der Agent selbst	Agent nicht anwesend	keine automatische

Der Weg vom Auslöser zum Lauf:



Benachrichtigung. Benachrichtigungen an Menschen und Agenten laufen über dieselbe Funktion:

1. Der Empfänger muss existieren und darf nicht gesperrt sein. Der Text wird auf 1500 Zeichen begrenzt; das ist die Länge des Textfelds im Messenger, dessen Datensätze eine feste Länge haben (Konzeptpapier „Aufbau der Plattform PWNC“, Abschnitt 4.2).
2. Hat der Empfänger Desktop-Zugang, legt PWNC die Nachricht in seinem Messenger ab, bei Desktop-Zugang des Absenders auch eine Kopie bei diesem, und setzt eine Ungelesen-Markierung.

3. Eskaliert wird nur, wenn der Empfänger nicht anwesend ist. Bei Konten, die keine Plattform-Agenten sind, darf außerdem keine Ungelesen-Markierung aus einer früheren Benachrichtigung bestanden haben.
4. Bei einem Plattform-Agenten startet die Eskalation ein neues Gespräch mit dem Agenten als Eigentümer. Die Nachricht ist der Text der Benachrichtigung. Im Assistentenmodus folgen die Adresse des Messengers, sofern die Nachricht dort abgelegt wurde, und der Hinweis auf den Assistentenmodus in der Sprache des Kontos.
5. Bei anderen Konten, und bei einem Plattform-Agenten, dessen Lauf nicht startet, versendet sie eine E-Mail, wenn eine Adresse hinterlegt ist: mit der Adresse des Desktops oder, ohne Messenger-Zugang, mit dem Text.

Jede Benachrichtigung an einen nicht anwesenden Plattform-Agenten startet ein eigenes Gespräch; die Ungelesen-Markierung hält Agenten nicht zurück. Ein Agent im Assistentenmodus liest den Messenger des Kontos über die mitgelieferte Adresse mit `get`. Das Öffnen des Messengers löscht die Markierung.

Ereignisse. Ein Ereignis (Backend-Anwendung „Ereignisse“) hat einen Namen, Empfänger, einen Nachrichtentext mit den Platzhaltern `%name%`, `%application%`, `%url%`, `%time%` und `%user%`, Bedingungen und einen Zeitplan.

- **Bedingungen** gelten für die aktuelle Anfrage: Adresse, Anfrageparameter (`name=muster`), „Ausgabe enthält“ und „Ausgabe enthält nicht“ (Textausgaben bis 1 MB). Muster erlauben `*` als Platzhalter und `!` als Verneinung; PWNC übersetzt sie in reguläre Ausdrücke. Je Feld genügt eine zutreffende Zeile, gesetzte Felder müssen alle zutreffen.
- **Zeitpläne** bestehen aus Zeitpunkten (`2026-11-01 08:00`), Intervallen (`1d2h30m`) und Cron-Ausdrücken. Der nächste Fälligkeitszeitpunkt liegt je Ereignis in einer Datei, die beim Prüfen gesperrt wird, damit gleichzeitige Anfragen ein Ereignis nicht doppelt auslösen.
- **Prüfung:** Am Ende jeder Anfrage außerhalb des Arbeitsprozesses prüft PWNC alle aktiven Ereignisse, zuerst die Fälligkeit, sonst die Bedingungen. MCP-Werkzeuge außer `get` und `post` enden vor diesem Schritt.
- **Auslösung:** Je Auslösung entsteht eine Hintergrundaufgabe; gesperrte Empfänger überspringt sie. Einem Plattform-Agenten ohne Assistentenmodus startet sie ein neues Gespräch mit dem Nachrichtentext. Eigentümer und Absender ist das Konto `trigger`; eine Messenger-Nachricht entsteht nicht, und die Anwesenheit spielt keine Rolle. Allen anderen Empfängern, auch Agenten im Assistentenmodus, schickt sie eine Benachrichtigung mit dem Absender `trigger`. Nach dem Einreihen stößt PWNC den Hintergrund an, sodass die Aufgabe ohne weitere Anfrage startet (4.10).

Zeitpläne werden nur bei Anfragen geprüft; ein fälliges Ereignis wird mit der ersten Anfrage nach seinem Zeitpunkt ausgelöst. Weil auch die Seitenabrufe von Agenten geprüft werden, kann die Arbeit eines Agenten Ereignisse auslösen, die andere Agenten starten.

Externe Agenten sind über MCP nicht erreichbar, weil dort der Client fragt und PWNC antwortet. Benachrichtigungen erreichen ein externes Agentenkonto wie einen Menschen: im Messenger, den es selbst abrufen muss, oder per E-Mail. Antworten auf eigene Aufrufe findet es als Wiedervorlage im Gedächtnis.

4.8 Aufrufe

Ein Aufruf ist eine Nachricht an ein anderes Konto mit Rückweg für die Antwort. Das Werkzeug `call` erwartet ein Ziel (`user.{name}`), die Nachricht und optional eine Kontext-ID.

Ablauf beim Aufrufer.

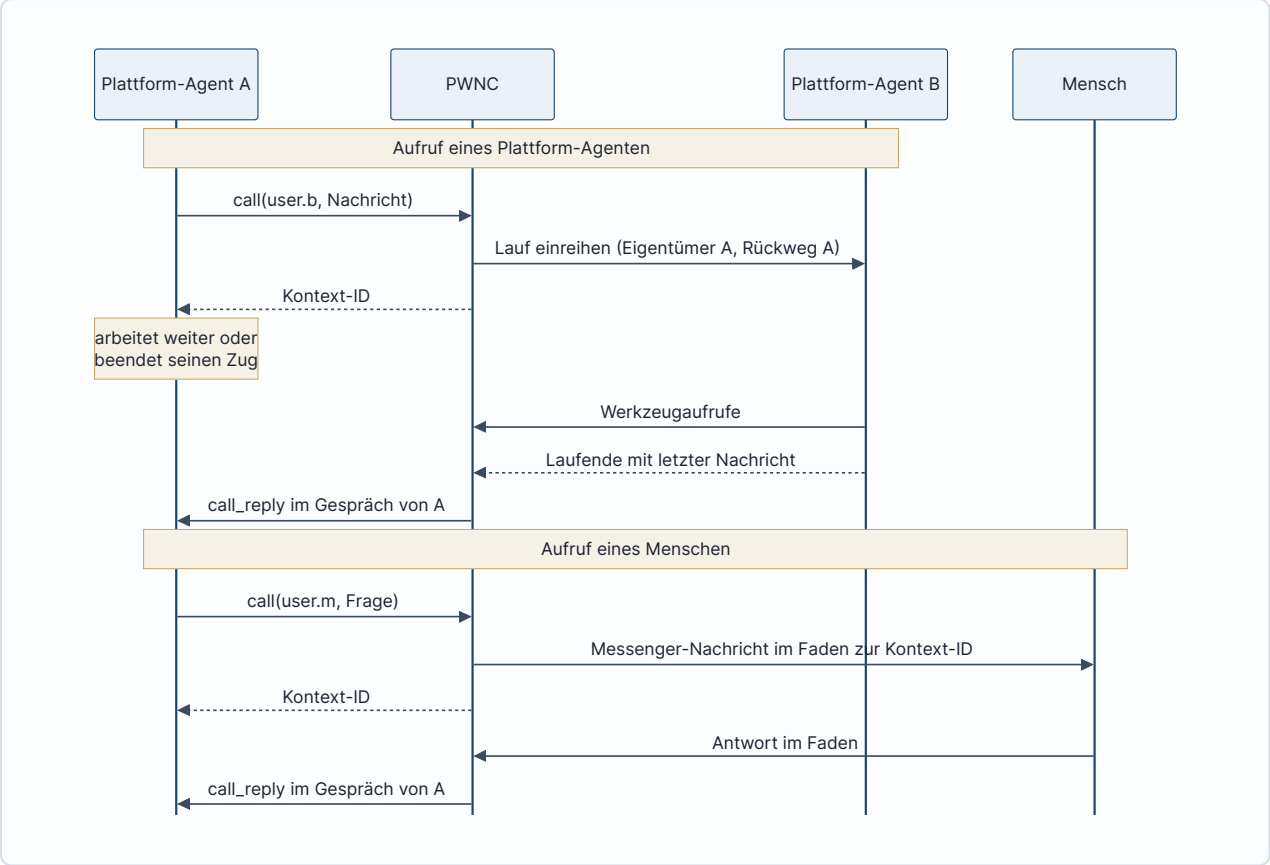
1. PWNC prüft das Ziel: vorhandenes Konto, nicht gesperrt, nicht das eigene.
2. Ist das Ziel ein Plattform-Agent und nicht anwesend, ist es ein Agentenaufruf; der Aufrufer braucht dafür `agent.{name}`. Sonst ist es ein Benutzeraufruf.
3. Kontext-ID: `false` beginnt einen neuen Austausch. Ohne Angabe übernimmt ein Plattform-Agent als Aufrufer seine eigene Kontext-ID; ein externer Aufrufer erhält eine neue. Eine angegebene ID setzt den Austausch mit dieser ID fort.
4. Das Werkzeug antwortet sofort mit der Kontext-ID; die Antwort des Ziels folgt später. Der Aufrufer wartet also nicht, während das Ziel arbeitet.

Agentenaufruf. PWNC beginnt oder setzt das Gespräch `{aufrufer}/{ziel}/{kontext-id}` fort und startet den Lauf des Ziels mit dem Aufrufer als Eigentümer und Rückweg. Endet der Lauf, geht seine letzte Textnachricht an den Aufrufer.

Benutzeraufruf. Die Nachricht ist auf 1500 Zeichen begrenzt. PWNC leitet aus der Kontext-ID einen Messenger-Faden ab und schickt eine Benachrichtigung (4.7). Wurde sie im Messenger zugestellt und hat der Aufrufer Desktop-Zugang, registriert PWNC einen Rückweg für diesen Faden. Jede Antwort des Empfängers in diesem Faden an den Aufrufer wird dann weitergeleitet. Ohne Rückweg ist die Nachricht eine Einbahnnachricht, und das Werkzeug teilt das mit.

Zustellung der Antwort.

- An einen Plattform-Agenten: PWNC schreibt einen erzeugten Aufruf `call_reply` und dessen Ergebnis in das Gespräch des Aufrufers, entweder „`ziel` (Name) answers your call: …“ oder, bei einem Fehler oder ohne Antworttext, „could not answer your call“ mit dem Stoppgrund. Ist für dieses Gespräch eine Aufgabe eingereiht oder aktiv, übernimmt sie die Einträge vor ihrer nächsten Modellanfrage; sonst wird ein neuer Lauf eingereiht.
- An einen externen Agenten: PWNC legt in dessen persönlicher Region einen Eintrag mit dem Stichwort `callback`, der Kontext-ID als Beschreibung und der Antwort als Inhalt an (bei Fehler oder ohne Antworttext: „{Stoppgrund} (no reply)“), sofort fällig. Der Agent sieht ihn bei seinem nächsten Gedächtnisabruf als Wiedervorlage.



Regeln.

- Ein Agentenaufruf liefert eine Antwort: die letzte Textnachricht beim Ende des gerufenen Laufs; Text aus früheren Läufen desselben Gesprächs zählt nicht. Spätere Ergebnisse meldet der Gerufene mit einem eigenen Aufruf unter derselben Kontext-ID.
- Ein Lauf, der durch eine zugestellte Antwort neu beginnt, hat keinen Aufrufer. Sein Ergebnis erreicht einen früheren Aufrufer nur, wenn der Agent es mit `call` weitergibt.
- Läufe desselben Gesprächs laufen nacheinander. Der Aufrufer muss seinen Zug für die Antwort nicht beenden.
- Text, den der Aufrufer nach dem Aufruf schreibt, erreicht das Ziel nicht; das Ziel erreicht er nur über `call`.

4.9 Chat und Messenger

Chat. Der Chat ist die Backend-Anwendung „Agenten“ (`interface.agent`), die meist aus der Verwaltungsoberfläche in einem eigenen Fenster geöffnet wird; bei der Bearbeitung einer Seite öffnet ihn der Menüeintrag „KI-Agenten“. Verwalter des Chats haben zusätzlich das Recht `interface.agent.operator`; sie verwalten dort auch die externen MCP-Server (4.6) und die eigenen Skills (4.2). Wer mindestens einen Plattform-Agenten nutzen darf, findet dort die Auswertung (4.12).

- **Auswahl:** Ein Gespräch wird in drei Schritten gewählt: 1. **Eigentümer**, also wessen Gespräche: Benutzer sehen nur ihre eigenen. Verwalter können jeden Eigentümer wählen, auch Agenten, die Gespräche mit anderen Agenten führen. 2. **Agent:** Zur Wahl stehen die nicht gesperrten Plattform-Agenten, für die man das Recht `agent.{name}` hat. Verwalter sehen außerdem jeden Agenten, mit dem der gewählte Eigentümer Gespräche hat. 3. **Gespräch:** eines der 100 jüngsten Ge-

spräche mit diesem Agenten oder ein neues. Ein Gespräch erscheint unter der Zusammenfassung der zuletzt gemeldeten Aufgabe, ohne Meldung unter seiner Kontext-ID.

- **Senden:** Der Chat hängt den Seitenkontext des öffnenden Fensters an: dessen Adresse und versteckte Formularfelder, etwa Modul und gewähltes Objekt, ohne die als „ignore/omit“ markierten. Er sendet den Kontext nur, wenn er sich seit der letzten Nachricht geändert hat, damit ein unveränderter Kontext nicht mit jeder Nachricht erneut Platz im Gespräch belegt; ein Trennzeichen (U+E000) trennt ihn vom Text.
- **Anzeige:** Alle 5 Sekunden lädt der Chat neue Protokolleinträge: Systemprompt, Reasoning, Seitenkontext, Werkzeugaufrufe mit Eingabe, Ergebnis und beiden Größen, Nachrichten mit dem Tokenverbrauch je Zug, Stoppgründe.
- **Bewertung:** Meldet der Agent eine Aufgabe mit `task_completed`, fragt der Chat unter diesem Aufruf, ob der Agent die Aufgabe hinreichend erfüllt hat, mit den Antworten „Gut“ und „Schlecht“. Ein erneuter Klick auf die gewählte Antwort nimmt die Bewertung zurück. Bewerten kann jeder, der das Gespräch sehen kann (4.12).
- **Eingriff:** Unterbrechen kann jeder, der das Gespräch sehen kann; löschen können nur Verwalter.

Messenger. Der Messenger ist Teil des Desktops (Recht `desktop`). Nachrichten haben Faden, Absender, Empfänger und Status; jedes Konto mit Desktop-Zugang kann jedem anderen schreiben. Agenten bedienen ihn über `get` und `post` wie Menschen. Antworten in Fäden mit registriertem Rückweg leitet PWNC an den Aufrufer weiter (4.8). Das Öffnen des Messengers löscht die Ungelesen-Markierung (4.7).

4.10 Hintergrundverarbeitung

Warteschlange. Eine Aufgabe ist eine PHP-Datei in `data/#daemon/`, erzeugt durch `cms_daemon(code, kennung, intervall)`. Eine Aufgabe mit derselben Kennung wird nicht erneut eingereiht, solange sie aussteht oder läuft. Wiederkehrende Aufgaben werden erst nach Ablauf ihres Intervalls seit der letzten Ausführung wieder eingereiht. Eine Markierungsdatei zeigt an, dass Arbeit ansteht; jede Anfrage prüft so eine einzige Datei, statt die Warteschlange zu lesen.

Start eines Arbeitsprozesses. Jede Anfrage außerhalb des Arbeitsprozesses reiht am Ende der Initialisierung die fälligen Wartungsaufgaben ein und prüft dann die Markierung und einen freien Platz; dasselbe geschieht direkt nach dem Einreihen eines Laufs. Die Zahl der Arbeitsprozesse ist begrenzt, ohne Einstellung auf zwei; die Grenze steht in den Systemdaten (`daemon`, `proclimit`), eine Einstellung in der Oberfläche gibt es nicht. Ist beides gegeben, ruft PWNC `pwnc/module/daemon.php` per HTTP bei sich selbst auf und wartet höchstens eine Sekunde. Aufgaben, die eine Anfrage erst danach einreiht, etwa ausgelöste Ereignisse, stoßen den Hintergrund beim Einreihen selbst an.

Ausführung. Der Arbeitsprozess sammelt alle ausstehenden Aufgaben und ordnet sie nach Einreihungszeit, einmalige Aufgaben zuerst. Jede Aufgabe wird vor der Ausführung gesperrt, sodass nur ein Prozess sie ausführt. Warten danach noch ungesperrte Aufgaben, startet er einen weiteren Arbeitsprozess, sofern die Grenze es zulässt. Jede Aufgabe hat ein PHP-Zeitlimit von 600 Sekunden; unter Linux zählt dabei die Rechenzeit, nicht das Warten auf Netzwerkantworten. Ein Lauf, der lange auf den Anbieter wartet, wird dort deshalb nicht durch das Warten beendet. Erledigte einmalige Aufgaben werden gelöscht, wiederkehrende geleert. Eine Statusdatei hält die letzten 25 Meldungen.

Wartung. Diese Aufgaben reiht jede Anfrage ein, sobald ihr Intervall abgelaufen ist:

Aufgabe	Intervall
Cache-Bereinigung, einschließlich Agentengesprächen ohne Änderung seit 30 Tagen	1 Stunde
Protokollpflege	5 Minuten
Suchindex aktualisieren	5 Minuten
Suchbewertung berechnen	1 Stunde
Gedächtnisbereinigung	1 Stunde
Bereinigung der Auswertung (4.12)	1 Stunde

MCP-Werkzeuge außer `get` und `post` enden vor diesem Schritt.

Folge. Ohne Anfragen an die Plattform laufen keine Hintergrundaufgaben, und keine Zeitpläne werden geprüft. Ein Cron-Job, der regelmäßig eine gewöhnliche Seite abruft, ersetzt fehlende Anfragen. Ein direkter Aufruf von `daemon.php` führt nur bereits eingereihte Aufgaben aus.

4.11 Steuerung

Rechte. Rechte sind hierarchische Bezeichner wie `interface.agent` oder `memory.scope.docs.reader`, die Konten und Gruppen gewährt oder entzogen werden. `*` und `operator` gelten für alle Unterebenen. Am Konto und in den Gruppen geht jeweils der Entzug der Gewährung vor; eine Festlegung am Konto geht den Gruppen vor. Rechte des anonymen Benutzers gelten für alle.

Agentenbezogen sind `agent.{name}` (Chat mit einem Plattform-Agenten und Aufruf), `interface.agent` und `interface.agent.operator` (Chat, Verwaltung aller Gespräche, der externen MCP-Server und der eigenen Skills, Auswertung aller Agenten), `mcp.{name}` (Werkzeuge eines externen MCP-Servers), `skills.{datei}` (ein eigener Skill), die Gedächtnisrechte und `desktop` (Messenger). Alles andere regeln die Rechte der jeweiligen Anwendung. Ein Agent mit Verwaltungsrechten richtet Konten, Schlüssel und Ereignisse über die Oberfläche ein wie ein Mensch.

Eingriff. Unterbrechen setzt ein Signal, das der Lauf vor jeder Modellanfrage und vor jedem Werkzeugaufruf prüft; wartet der Lauf noch in der Warteschlange, wird die Aufgabe entfernt. Löschen während eines Laufs setzt zusätzlich ein Löschsignal; der Lauf löscht die Gesprächsdatei an seinem Ende.

Zugangsdaten. Schlüssel der Konten speichert PWNC nur als Hash. Die Schlüssel der KI-Anbieter und die Zugangsdaten externer MCP-Server liegen im Klartext in den Daten der Installation (`data/#system/permission.dat`, `data/#system/mcp.server.dat`). Ein Hash genügt hier nicht, weil PWNC sie im Klartext an Anbieter und Server senden muss; wer diese Dateien lesen kann, kann sie verwenden. Im Connector-Modus erhält der Anbieter die Zugriffstoken der externen Server mit jeder Anfrage.

Nachvollziehbarkeit. Das Protokoll enthält Systemprompt, MCP-Endpunkt, Nachrichten, Anfragen ohne Nachrichtenliste und Zugriffstoken, Rohantworten, Werkzeugaufrufe mit gekürzter Eingabe, Werkzeugergebnisse, Kompaktierungen und Stopps. Der Chat zeigt den Tokenverbrauch je Zug und insgesamt. Agenten wird ihr Verbrauch nicht mitgeteilt: Kennen sie ihre Grenzen, kürzen sie ihre Arbeit ab, und die Qualität sinkt. Zudem würde die Angabe selbst Tokens kosten und im Systemprompt den Prompt-Cache aufheben.

4.12 Auswertung

PWNC erfasst die Arbeit von Plattform-Agenten je Aufgabe und stellt sie in einer Ansicht je Agent dar. Die Kennzahlen stammen aus dem Protokoll, die Bewertungen von Menschen.

Aufgabe. Eine Aufgabe umfasst die Protokolleinträge seit der vorigen und endet, wenn der Agent `task_completed` mit einer Zusammenfassung von höchstens 100 Zeichen aufruft; die Schlussantwort desselben Laufs zählt noch dazu. Das Werkzeug gibt es nur in Läufen. Jede Modellantwort mit einer Meldung schließt eine eigene Aufgabe ab; meldet eine Antwort mehrmals, entsteht eine Aufgabe mit der letzten Zusammenfassung. Die Werkzeugbeschreibung verlangt nur die Meldung; dass Aufgaben bewertet werden, erfahren Agenten nicht.

Erfassung. Die Tabelle `cms_agent_eval` hat eine Zeile je Aufgabe, gekennzeichnet durch Agent, Kontext-ID und die Byte-Position im Protokoll, an der die Aufgabe beginnt. Sobald das Ergebnis einer Meldung im Protokoll steht, und am Ende jedes Laufs, außer nach einem Abbruch des Prozesses, liest PWNC die neuen Protokolleinträge und schreibt fort:

- Modellanfragen, davon Wiederholungen nach vorübergehenden Fehlern (4.5),
- ausgeführte Werkzeugaufrufe, davon fehlgeschlagene; nicht gezählt werden Aufrufe, die eine Unterbrechung beendet hat, und die erzeugten Aufrufe `call_reply` (4.8),
- fehlgeschlagene Läufe,
- die Antwortzeit der Modellanfragen von der Anfrage bis zur Antwort, in ganzen Sekunden; Wartezeiten vor Wiederholungen zählen nicht, die Ausführung von Werkzeugen nur im Connector-Modus, in dem der Anbieter sie während seiner Antwort ausführt (4.5),
- den Tokenverbrauch nach Eingabe, Cache und Ausgabe.

Dazu kommen Herkunft (Chat, Ereignis, Aufruf oder Benachrichtigung), Status (offen, erledigt, unterbrochen oder fehlgeschlagen) und Zusammenfassung. Eine Unterbrechung durch einen Menschen gilt nicht als Fehlschlag, das Erreichen der Anfragegrenze schon. Zwei Stopps zählen nicht, weil sie nicht beim Agenten liegen: ein Abbruch des Prozesses, nach dem der Lauf fortsetzt, und ein Fehler, bevor der Lauf eine Modellanfrage gestellt hat. Jeder andere Stopp setzt den Status, bis die Aufgabe gemeldet ist.

Bewertung und Aufbewahrung. Erledigte Aufgaben bewerten Menschen im Chat als gut oder schlecht (4.9). Jede Fortschreibung setzt den Zeitpunkt der Aufgabe neu, eine Bewertung nicht. Die stündliche Bereinigung löscht Aufgaben, deren letzte Fortschreibung länger zurückliegt als die eingestellte Frist, ohne Einstellung 90 Tage. Wird ein Konto gelöscht, entfällt seine Auswertung.

Ansicht. Den Menüeintrag „Auswertung“ im Chat sieht, wer mindestens einen Plattform-Agenten nutzen darf (`agent.{name}`). Die Ansicht zeigt die nicht gesperrten Agenten mit Aufgaben im Zeitraum, für die der Betrachter dieses Recht hat; Verwalter des Chats sehen alle. Zeitraum sind so viele Kalendertage bis einschließlich heute, wie die Frist Tage hat. Die Tage richten sich nach der Zeitzone des eigenen Kontos, und eine Aufgabe zählt zum Tag ihrer letzten Fortschreibung. Je Agent zeigt eine Zeile sechs Kennzahlen:

Kennzahl	Berechnung
Aufgaben	Zahl der Aufgaben
Zuverlässigkeit	Mittel über die Aufgaben mit mindestens einer Modellanfrage. Eine Aufgabe zählt 0, wenn sie fehlgeschlagen ist oder wenn sie aus einem Ereignis, einem Aufruf oder einer Benachrichtigung stammt und offen ist, sonst 1 − (Wiederholungen + fehlgeschlagene Läufe) / Modellanfragen

Kennzahl	Berechnung
Tool-Kompetenz	1 − fehlgeschlagene / ausgeführte Werkzeugaufrufe
Effektivität	Anteil der Bewertungen „Gut“ an allen Bewertungen, daneben die Zahl der Bewertungen
Ø Dauer/Schritt	Antwortzeit / Modellanfragen, in Sekunden
Ø Tokens/Schritt	Ausgabetokens / Modellanfragen

Zuverlässigkeit, Tool-Kompetenz und Effektivität stehen in Prozent. Eine offene Aufgabe aus dem Chat zählt nach ihren Anfragen, weil ein Gespräch dort ohne Meldung weitergehen kann; aus Ereignissen, Aufrufen und Benachrichtigungen zählt sie wie eine fehlgeschlagene. Dauer und Tokens je Schritt berücksichtigen nur Aufgaben ohne Wiederholung und ohne Fehlschlag, die offen oder erledigt sind; in diesen Aufgaben ist jede Modellanfrage ein Arbeitsschritt des Agenten. Gezählt werden nur Ausgabetokens: Sie erzeugt das Modell selbst, während die Eingabe mit dem Gespräch wächst und vom Caching des Anbieters abhängt.

Die Werte einer Zeile entstehen aus den Summen über den ganzen Zeitraum. Lässt sich eine Kennzahl nicht berechnen, etwa ohne Bewertung, steht „–“. Die Werte sind absolut. Eine Gesamtnote gibt es nicht, weil ihre Gewichtung willkürlich wäre und Agenten mit fehlenden Kennzahlen nicht vergleichbar wären; einen Bezug auf andere Agenten auch nicht, weil sich etwa ein Median mit jedem Agenten ändert. Die Tabelle lässt sich stattdessen nach jeder Kennzahl und nach dem Namen des Agenten auf- und absteigend sortieren, „–“ gilt dabei als kleinster Wert. Die Sortierung bleibt je Konto gespeichert; voreingestellt ist die Zuverlässigkeit, absteigend. Der Name eines Agenten führt zu den eigenen Gesprächen mit ihm.

Verlauf. Unter jeder Zeile zeigt ein Diagramm die Kennzahlen je Tag über den Zeitraum: eine Linie je Kennzahl in der Farbe ihres Spaltenkopfs, dazu senkrechte Linien an den Montagen. Angezeigt wird die Kennzahl, nach der die Tabelle sortiert ist, bei der Sortierung nach dem Namen alle; weitere lassen sich über ein Feld im Spaltenkopf für alle Diagramme ein- und ausblenden, damit sich die Linien nicht überdecken. Eine gestrichelte Linie in derselben Farbe zeigt den Wert der Tabelle über den ganzen Zeitraum, bei den Aufgaben den Durchschnitt je Tag mit Aufgaben. Die Tageswerte entstehen nach denselben Formeln aus den Aufgaben des Tages. Tage ohne Wert überspringt eine Linie; sie beginnt am Vortag ihres ersten Werts bei null. Zwischen den Werten verläuft sie als monotone kubische Kurve nach Steffen [44]: Zwischen zwei benachbarten Werten verläuft sie monoton und schwingt nicht über sie hinaus; Höhen und Tiefen liegen nur an gemessenen Werten. Prozentwerte haben die Skala 0 bis 100 %. Für Aufgaben, Dauer und Tokens reicht die Skala bis zum 95. Perzentil der Tageswerte aller angezeigten Agenten, aufgerundet auf das 1-, 2-, 4-, 6-, 8- oder 10-Fache einer Zehnerpotenz; höhere Werte enden am oberen Rand. Ein einzelner Ausreißer staucht die Skala so nicht, und alle Diagramme haben dieselbe Skala. Zeigt der Mauszeiger auf einen Tag, erscheinen Wochentag, Datum und die Werte dieses Tages.

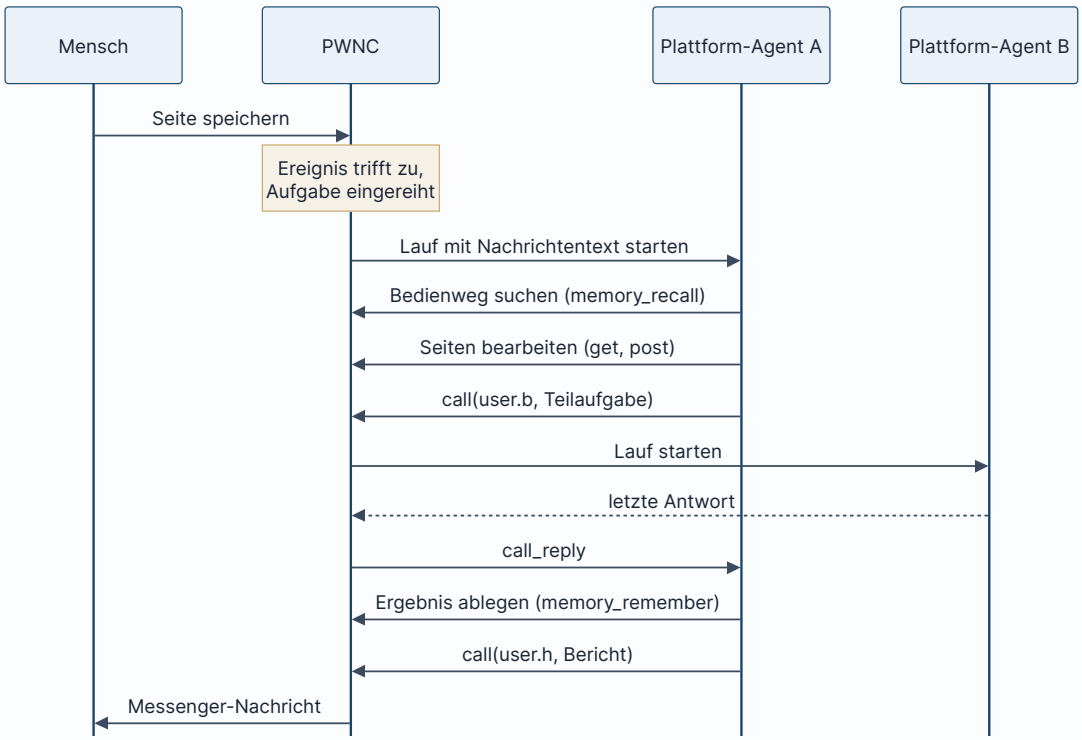
Kapitel 5

Zusammenspiel

Die Bausteine greifen an festen Stellen ineinander:

- **Eine Rechteprüfung.** Seiten, Gedächtnis, Skills, Aufrufe, Chat und Auswertung prüfen dieselben Rechte desselben Kontos. Für Agenten gibt es keinen Sonderweg. Bei externen MCP-Servern entscheidet das Recht `mcp.{name}`, welche Konten die einmal hergestellte Autorisierung nutzen.
- **Agentenanfragen sind Anfragen.** `get` und `post` durchlaufen den vollständigen Seitenaufbau einschließlich Ereignisprüfung und Hintergrundstart. Agenten lösen dieselben Ereignisse aus wie Menschen.
- **Ein Zustellweg.** Messenger-Nachrichten, Aufrufe an Menschen und Ereignisse laufen über dieselbe Benachrichtigung; nur Ereignisse an Plattform-Agenten ohne Assistentenmodus starten den Lauf direkt. Ob aus einer Benachrichtigung ein Lauf oder eine E-Mail wird, entscheiden Kontotyp und Anwesenheit, bei E-Mails zusätzlich die Ungelesen-Markierung.
- **Das Protokoll ist der Zustand.** Jeder Lauf baut das Gespräch aus dem Protokoll neu auf. Fortsetzung nach Abbruch, Nachrichten während eines Laufs und zugestellte Antworten nutzen denselben Weg: einen Eintrag im Protokoll und einen eingereihten Lauf. Auch die Auswertung liest ihre Kennzahlen aus dem Protokoll.
- **Ansicht und Gedächtnis.** Die Ansicht verringert die Datenmenge je Abruf, das Gedächtnis die Zahl der Abrufe für bereits bekannte Bedienwege.
- **Templates.** Der Laufcode liest nur Template-Felder; welcher Anbieter verwendet wird, ist eine Angabe am Konto.

Ein Ablauf über mehrere Bausteine:



Kapitel 6

Einordnung gegenüber anderen Systemen

Grundlage ist eine Recherche in Dokumentationen, Spezifikationen und Fachveröffentlichungen mit Stand vom 23. Oktober 2026; die Anbindung externer MCP-Server ist mit Stand vom 31. Oktober 2026 ergänzt, die Auswertung mit Stand vom 16. November 2026. „Ohne direktes Vorbild“ heißt: Die Recherche fand kein System, das den Mechanismus in dieser Form umsetzt. Das schließt nicht aus, dass es eines gibt.

Die meisten Bausteine entsprechen Lösungen, die auch andere Systeme verwenden (6.1); Übereinstimmung heißt dabei nicht Übernahme. Sieben weichen von diesen Lösungen in einem wesentlichen Punkt ab (6.2). Für neun Mechanismen fand die Recherche kein direktes Vorbild (6.3). Sie betreffen die Bedienung über die Oberfläche, den Austausch zwischen Agenten und Menschen, den Umgang mit dem Gedächtnis und den Betrieb ohne eigene Infrastruktur.

6.1 Gleiche Lösungen in anderen Systemen

Mechanismus in PWNC	Vergleichbare Systeme
Agent als Benutzerdatensatz im Rechtesystem der Anwendung	ServiceNow: „AI user“ als Datensatz der Benutzertabelle [1]; Microsoft Entra Agent ID: Benutzerkonto als Untertyp, mit Postfach und Teams über Agent 365 [2]
Gedächtnis in Bereichen mit Zugriffsrechten	Google Memory Bank: IAM-Bedingungen je Scope [3]; Claude Managed Agents: Speicher je Sitzung lesend oder schreibend eingebunden [4]; Letta: von mehreren Agenten geteilte Blöcke [5]
Abruf in Stufen, Inhalte erst auf Anforderung (Gedächtnis, Skills)	Anthropic Agent Skills: Beschreibungen zuerst, Inhalte bei Bedarf [6]; MemGPT: Speicherhierarchie, durch die das Modell blättert [7]
Löschen ungenutzter Erinnerungen	GitHub Copilot Memory: Löschung nach 28 Tagen ohne Nutzung [8]; MemoryBank: Vergessenskurve [9]; ExpeL: Bewertungen durch den Agenten, Löschung bei Zählerstand 0 [10]
Ersetzen alter Werkzeugergebnisse bei Kontextknappheit	Anthropic Context Editing [11]
Gesprächsprotokoll als Datei, Fortsetzung daraus	Claude Agent SDK: Sitzungen als JSONL [12]; LangGraph: Checkpointer [13]
Anbieterangaben als Konfiguration	LiteLLM: JSON-Einträge für OpenAI-kompatible Anbieter [14]
Werkzeugausführung beim Anbieter über MCP, auch für externe Server mit deren Token	Anthropic MCP-Connector [15]; OpenAI Responses API [16]
Anbindung externer MCP-Server mit erkannter Autorisierung: keine, fester Schlüssel oder OAuth mit Erkennung und Registrierung	MCP-Spezifikation 2026-07-28, Autorisierung [39]; Microsoft Copilot Studio: keine Authentifizierung, API-Schlüssel oder OAuth 2.0 als „Dynamic discovery“, „Dynamic“ oder „Manual“ [40]
Differenzen statt vollständiger Beobachtungen	Playwright MCP: inkrementelle Snapshots seit Version 0.0.43 [17]; Studien zu Differenzverläufen [18] [19]
Fortsetzbare Austausche über Kontext-IDs	A2A: `contextId` und `taskId` [20]

Mechanismus in PWNC	Vergleichbare Systeme
Hintergrundaufgaben über Anfragen an die eigene Installation	WordPress WP-Cron [21]
Start von Agenten durch Ereignisse und Zeitpläne	Microsoft Copilot Studio: Ereignis- und Wiederholungsauslöser [22]
Agentenläufe, die nach einer Unterbrechung fortsetzen	Cloudflare Agents: gespeicherte Zustände und Zwischenstände [23]
Abschluss einer Aufgabe durch ein Werkzeug des Agenten	Cline: `attempt_completion` [50]; AutoGPT: `finish` mit Zusammenfassung [52]; OpenHands: `finish` mit Selbsteinschätzung `task_completed` [53]
Bewertung durch Menschen mit Daumen hoch oder runter	Copilot Studio: Reaktionen auf einzelne Antworten, mit Verlauf über die Zeit [45]; Langfuse: Rückmeldungen als Scores an Traces [47]; Cline 3.17: Schaltflächen an der Abschlussmeldung [51]
Kennzahlen aus der Ausführung: Anfragen, Antwortzeit, Tokens, Werkzeugaufrufe und ihre Fehler	LangSmith: Dashboards zu Latenz, Fehlerquoten, Tokens und Werkzeugaufrufen [46]; OpenTelemetry: Metriken für Dauer sowie Modell- und Werkzeugaufrufe je Agentenaufruf, Status „Development“ [48]
Übersicht je Agent in der Plattform, in der die Agenten arbeiten	Microsoft Foundry: Agententabelle mit Läufen, Fehlerquote und Tokenverbrauch [49]; Copilot Studio: Ergebnisse, Reaktionen und Erfolg der Werkzeuge je Agent, mit Verlauf [45]

6.2 Abweichungen

Agent ohne Sonderstatus. Bei Microsoft ist das Benutzerkonto eines Agenten ein Untertyp: ohne Passwort, ohne interaktive Anmeldung, an eine Agentenidentität gebunden und von privilegierten Administratorrollen ausgeschlossen [2]. ServiceNow kennzeichnet den Datensatz als „AI user“ [1]. PWNC kennt keinen Untertyp; ein Mensch kann dasselbe Konto im Browser nutzen, im Assistentenmodus ausdrücklich als Inhaber neben dem Agenten, und ein Agent kann jedes Recht erhalten. Folge: Eine Rechteverwaltung gilt für alle Konten. Einschränkungen für Agenten entstehen nur über die vergebenen Rechte, nicht über den Kontotyp.

Gedächtnisrechte aus dem Rechtesystem der Plattform. Google regelt den Zugriff über IAM-Bedingungen seiner Cloud [3], Claude Managed Agents über die Einbindung je Sitzung [4], Letta über die Zuordnung von Blöcken zu Agenten [5]. In PWNC sind Gedächtnisrechte gewöhnliche Rechte der Plattform und gelten für jedes Konto, das über MCP zugreift; Regionen entstehen beim ersten Schreiben, und einzelne Einträge tragen eigene Schutzkennzeichen. Folge: Gedächtnis- und Anwendungsrechte werden an einer Stelle verwaltet. Eine Gedächtnisanwendung in der Browser-Oberfläche und eine Versionsgeschichte der Einträge wie bei Claude Managed Agents [4] gibt es nicht.

Templates mit Ablaufsteuerung. LiteLLM beschreibt OpenAI-kompatible Anbieter in JSON; komplexe Umwandlungen und Änderungen am Werkzeugaufruf verlangen Python-Code [14]. PWNC beschreibt im Template auch Werkzeugformate, die Zuordnung von Stoppgründen zu Aktionen und das Muster für Kontextüberläufe. Folge: Der Laufcode bleibt anbieterunabhängig. Was das Template-Schema nicht abbildet, erfordert eine Änderung des Laufcodes.

Eine Autorisierung je externem Server. In Copilot Studio wählt der Ersteller je Werkzeug, ob es mit den Anmeldedaten des jeweiligen Benutzers oder mit seinen eigenen arbeitet; voreingestellt sind die des Benutzers, die eigenen empfiehlt Microsoft für gemeinsam genutzte Ressourcen [41]. PWNC kennt nur

die zweite Form: Ein Verwalter stellt die Autorisierung einmal her, und alle Konten mit `mcp.{name}` nutzen sie. Folge: Beim externen Dienst laufen alle Zugriffe unter der Identität, mit der verbunden wurde. Welche Agenten zugreifen, regeln die Rechte in PWNC; welche Daten erreichbar sind, regeln die Rechte dieser Identität beim Dienst.

Aufgabe als Einheit der Auswertung, abgegrenzt durch den Agenten. Copilot Studio wertet Sitzungen aus: Eine Sitzung gilt als „Resolved confirmed“, wenn das Thema „End of Conversation“ ausgelöst wird und der Benutzer den Erfolg bestätigt, und als „Abandoned“, wenn sie nach 30 Minuten abläuft, ohne gelöst oder eskaliert worden zu sein [45]. LangSmith und Langfuse ordnen Kennzahlen und Rückmeldungen Traces zu [46] [47]. Dass ein Agent seine Aufgabe mit einem eigenen Werkzeug abschließt, ist verbreitet (6.1); bei Cline 3.17 trägt die Abschlussmeldung Schaltflächen für eine Rückmeldung [51]. In PWNC grenzt die Meldung die Einheit der Auswertung ab: Die Kennzahlen aus dem Protokoll und die Bewertung hängen an ihr, und die Ansicht fasst sie je Agent zusammen. Folgen: Die Einheit braucht keine Konfiguration und entspricht dem, was der Agent als erledigt meldet; wie sauber sie Aufgaben trennt, hängt davon ab, ob der Agent meldet (7). Ob die Aufgabe gelungen ist, sagt die Meldung nicht. Bei OpenHands traf die Selbsteinschätzung des Agenten beim Abschluss nach Angabe des Autors der Änderung in rund 60 % der Fälle zu [53]; in PWNC entscheidet darüber die Bewertung durch Menschen.

Keine Bewertung durch ein Sprachmodell. Microsoft Foundry und Amazon Bedrock AgentCore bieten Bewerter, die ein Sprachmodell als Richter einsetzen, etwa für die Fragen, ob ein Agent eine Aufgabe erfüllt oder die richtigen Werkzeuge mit korrekten Parametern aufgerufen hat [54] [55]; Copilot Studio beurteilt die Qualität von Antworten mit KI [45]. PWNC zählt nur, was das Protokoll enthält, und überlässt das Urteil über das Ergebnis Menschen. Die Tool-Kompetenz zählt Aufrufe, die mit einem Fehler enden, auch bei Argumenten, die PWNC gegen das Schema abweist (4.2); sie liegt damit näher an Foundrys „Tool Call Success“ als an der „Tool Call Accuracy“, die auch Wahl und Parameter der Aufrufe beurteilt [54]. Folgen: Die Auswertung verursacht keine Modellkosten und hängt nicht vom Urteil eines Modells ab. Über unbewertete Aufgaben sagt sie nur, ob sie technisch reibungslos liefen.

Zuverlässigkeit im Betrieb. Foundry zeigt je Agent als Fehlerquote das Verhältnis fehlgeschlagener zu erfolgreichen Läufen im letzten Monat [49]. τ-bench misst Zuverlässigkeit als Wahrscheinlichkeit, dass k Versuche derselben Aufgabe alle gelingen (pass^k) [56]. PWNC misst sie im Betrieb je Aufgabe: Wiederholungen nach vorübergehenden Fehlern und fehlgeschlagene Läufe im Verhältnis zu den Modellanfragen, dazu offene Aufgaben aus Ereignissen, Aufrufen und Benachrichtigungen als Fehlschlag (4.12). Folgen: Auch Störungen, die ein Lauf übersteht, senken den Wert. Ob ein Agent dieselbe Aufgabe bei Wiederholung gleich gut erledigt, misst er nicht.

6.3 Ohne direktes Vorbild

Gerenderte Oberfläche als vollständige Agentenschnittstelle. Verbreitet sind ein Werkzeug je Funktion, etwa je Endpunkt bei FastAPI-MCP [24]; allgemeine Werkzeuge über ausdrücklich freigegebene Fähigkeiten, etwa die drei Werkzeuge des WordPress-MCP-Adapters [25]; Seiten, die eigene Werkzeuge veröffentlichen, bei WebMCP auch aus annotierten Formularen [26]; und Agenten, die die Oberfläche im Browser bedienen [17]. Hypermedia als Schnittstelle für Agenten wird in Forschung und Fachpresse vorgeschlagen [27] [28]. PWNC setzt das auf dem Server um: `get` und `post` führen dieselben Links und Formulare aus wie für Menschen, im selben Prozess und unter der Identität des Agenten; die Ausgabe wird ohne Sprachmodell gefiltert. Die Recherche fand keinen MCP-Server, der die gesamte Oberfläche einer Anwendung so bereitstellt. Folgen: Neue Funktionen stehen Agenten ohne eigene

Werkzeuge zur Verfügung. Jede Aktion kostet dafür Seitenabrufe; auf dem Testsystem WebArena erzielten Agenten mit API-Zugriff bessere Ergebnisse als reine Browser-Agenten und eine Kombination aus beidem die besten [29].

Differenzen über Adressen hinweg. Playwright MCP fasst unveränderte Teile des Barrierefreiheitsbaums zusammen [17]. Studien empfehlen Differenzen für den Beobachtungsverlauf von Web-Agenten [19] und zeigen, dass Differenzen Eingaben verlängern, wenn sich Beobachtungen stark ändern [18]. PWNC bildet Unified Diffs der gefilterten HTTP-Antwort beliebiger Adressen auf dem Server, greift nach Formularversand und Weiterleitung auf die Antwort einer anderen Adresse zurück und liefert einen Diff nur, wenn er kürzer als 90 % der vollständigen Antwort ist. Für den Rückgriff auf andere Adressen und diese Kombination fand die Recherche kein Vorbild.

Ein Aufruf für Agenten und Menschen. A2A verbindet Agenten untereinander [20]. In MCP kann der Server den Benutzer des verbundenen Clients befragen [30]. Twilio A2H stellt Agenten einen eigenen Kanal zu Menschen bereit [31], Magentic-UI bindet den Menschen in Planung und Ausführung ein [32]. In PWNC entscheidet sich erst beim Aufruf, ob er einen Lauf startet oder eine Messenger-Nachricht erzeugt, auch danach, ob ein Mensch das Agentenkonto gerade nutzt. Menschen antworten im gewöhnlichen Messenger; der Rückweg richtet sich nach der Art des Aufrufers. Für die Weiterleitung nach Anwesenheit fand die Recherche kein Vorbild.

Gedächtnis als Rückkanal für zustandslose MCP-Clients. Seit MCP 2026-07-28 antwortet ein Server nur auf Anfragen des Clients; Benachrichtigungen beschränken sich auf Listen- und Ressourcenänderungen über einen Kanal, den der Client offen hält, und lange Vorgänge fragt der Client über die Task-Erweiterung ab [30]. A2A stellt Antworten an einen Webhook des Clients zu [20]. PWNC legt Antworten auf Aufrufe externer Agenten als fällige Einträge in deren persönlichem Gedächtnis ab, wo sie bei jedem Abruf obenan stehen, bis sie gelesen sind. Kein direktes Vorbild gefunden. Folge: Eine Antwort erreicht den externen Agenten erst bei seinem nächsten Gedächtnisabruf.

Lebensdauer abhängig von der Bewertung. Einzelteile haben Vorbilder: ExpeL löscht Erkenntnisse nach Abwertungen durch den Agenten [10], GitHub Copilot löscht Erinnerungen nach 28 Tagen ohne Nutzung [8], MemoryBank folgt einer Vergessenskurve [9], Generative Agents gewichten Erinnerungen nach Wichtigkeit, ohne sie zu löschen [33]. In PWNC ist die durchschnittliche Bewertung aller Bewerter der Faktor zweier Fristen, seit der letzten Änderung und seit dem letzten Lesen; gelöscht wird erst, wenn beide abgelaufen sind. Für diese Regel fand die Recherche kein Vorbild.

Stichworte statt Inhalte. Verbreitet ist, Erinnerungsinhalte in den Kontext zu laden oder einen Index mitzugeben, etwa die Beschreibungen von Skills oder die Datei `MEMORY.md` in Claude Code [6 34]. PWNC ordnet jeder Nachricht an einen Plattform-Agenten mit Werkzeugen bis zu drei Stichworte aus dem gemeinsamen Vokabular zu, nach Leserechten gefiltert, mit einem Klassifikator nach dem Verfahren der N-Gramm-Textkategorisierung [35]. Der Agent entscheidet, ob er damit sucht. Kein direktes Vorbild gefunden; die Recherche war in diesem Punkt weniger umfangreich als in den übrigen.

Ereignisse auf der gerenderten Ausgabe. Agentenplattformen starten Agenten über Ereignisse aus angebundenen Diensten und über Zeitpläne [22]. Regeln über Adresse, Parameter und Antwortinhalt einer Anfrage kennen Web Application Firewalls, dort zum Blockieren und Protokollieren [36]. PWNC prüft Bedingungen auf dem eigenen HTTP-Verkehr einschließlich der gerenderten Ausgabe, auch bei Abrufen durch Agenten, und startet damit Agenten. Kein Vorbild gefunden. Folge: Ketten aus Agenten entstehen ohne Orchestrierung; eine Begrenzung solcher Ketten gibt es nicht (7).

Agentenläufe auf gewöhnlichem PHP-Hosting. Hintergrundaufgaben über Anfragen an die eigene Installation sind in PHP verbreitet [21]. Agentenläufe, die Unterbrechungen überstehen, setzen sonst eigene Laufzeitumgebungen voraus, etwa Cloudflare Agents [23]. PWNC führt Agentenläufe mit dem PHP-Muster aus und setzt sie aus dem Protokoll fort. Kein dokumentiertes Vorbild gefunden. Folgen: Zusätzliche Infrastruktur ist nicht nötig. Parallelität, Laufzeit und Reaktionszeit hängen vom Verkehr und von den Grenzen des Hostings ab.

Anweisungen durch den MCP-Server. Das MCP-Feld `instructions` überträgt Anweisungen einmal mit der Beschreibung des Servers. Wiederholte Erinnerungen fügen sonst Clients ein, etwa Claude Code als Systemerinnerung neben Werkzeugergebnissen [37]. In PWNC fügt der Server den Instruktionstext des Kontos in jedes n-te Werkzeugergebnis ein. Kein Vorbild gefunden. Folge: Das wirkt mit jedem Client; nach der Instruktionshierarchie gewichten Modelle Text in Werkzeugergebnissen niedriger als Systemanweisungen [38].

Quellen

1. ServiceNow: AI agent security implementation. https://www.servicenow.com/docs/r/intelligent-experiences/aia-security-implementation.html
2. Microsoft Learn: The agent's user account in Microsoft Entra Agent ID. https://learn.microsoft.com/en-us/entra/agent-id/agent-users
3. Google Cloud: Memory Bank, IAM conditions. https://docs.cloud.google.com/gemini-enterprise-agent-platform/scale/memory-bank/iam-conditions
4. Anthropic: Using agent memory (Managed Agents). https://platform.claude.com/docs/en/managed-agents/memory
5. Letta: Shared memory. https://docs.letta.com/guides/agents/multi-agent-shared-memory
6. Anthropic: Equipping agents for the real world with Agent Skills (2025). https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills
7. Packer u. a.: MemGPT: Towards LLMs as Operating Systems (2023). https://arxiv.org/abs/2310.08560
8. GitHub Docs: Copilot Memory. https://docs.github.com/en/copilot/concepts/agents/copilot-memory
9. Zhong u. a.: MemoryBank (2023). https://arxiv.org/abs/2305.10250
10. Zhao u. a.: ExpeL: LLM Agents Are Experiential Learners (2023). https://arxiv.org/abs/2308.10144
11. Anthropic: Context editing. https://platform.claude.com/docs/en/build-with-claude/context-editing
12. Anthropic: Work with sessions (Claude Agent SDK). https://code.claude.com/docs/en/agent-sdk/sessions
13. LangChain: LangGraph Persistence. https://docs.langchain.com/oss/python/langgraph/persistence
14. LiteLLM: Adding OpenAI-compatible providers. https://docs.litellm.ai/docs/contributing/adding_openai_compatible_providers
15. Anthropic: MCP connector. https://platform.claude.com/docs/en/agents-and-tools/mcp-connector
16. OpenAI: MCP servers. https://developers.openai.com/api/docs/guides/tools-remote-mcp
17. Microsoft: Playwright MCP, Release 0.0.43. https://github.com/microsoft/playwright-mcp/releases/tag/v0.0.43
18. Piterbarg, Pinto, Fergus: diff History for Neural Language Agents (2023). https://arxiv.org/abs/2312.07540
19. Enomoto u. a.: Read More, Think More: Revisiting Observation Reduction for Web Agents (2026). https://arxiv.org/abs/2604.01535
20. A2A Protocol Specification. https://a2a-protocol.org/latest/specification/
21. WordPress Plugin Handbook: Cron. https://developer.wordpress.org/plugins/cron/
22. Microsoft Learn: Event triggers overview (Copilot Studio). https://learn.microsoft.com/en-us/microsoft-copilot-studio/authoring-triggers-about
23. Cloudflare: Long-running agents. https://developers.cloudflare.com/agents/concepts/long-running-agents/
24. Tadata: FastAPI-MCP. https://github.com/tadata-org/fastapi_mcp
25. WordPress: MCP Adapter. https://github.com/WordPress/mcp-adapter
26. Chrome for Developers: WebMCP declarative API. https://developer.chrome.com/docs/ai/webmcp/declarative-api
27. Ciortea u. a.: A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web (AAMAS 2019). https://hal-emse.ccsd.cnrs.fr/emse-02070625
28. Wagner: HATEOAS: The API Design Style That Was Waiting for AI (Nordic APIs, 2025). https://nordicapis.com/hateoas-the-api-design-style-that-was-waiting-for-ai/
29. Song u. a.: Beyond Browsing: API-Based Web Agents (2024). https://arxiv.org/abs/2410.16464
30. Model Context Protocol, Specification 2026-07-28: Key Changes. https://modelcontextprotocol.io/specification/2026-07-28/changelog
31. Twilio: Introducing A2H (2026). https://www.twilio.com/en-us/blog/products/introducing-a2h-agent-to-human-communication-protocol
32. Mozannar u. a.: Magentic-UI: Towards Human-in-the-loop Agentic Systems (2025). https://arxiv.org/abs/2507.22358
33. Park u. a.: Generative Agents: Interactive Simulacra of Human Behavior (2023). https://arxiv.org/abs/2304.03442
34. Anthropic: How Claude remembers your project (Claude Code). https://code.claude.com/docs/en/memory
35. Cavnar, Trenkle: N-Gram-Based Text Categorization (1994). https://www.let.rug.nl/vannoord/TextCat/textcat.pdf

36. OWASP ModSecurity: Reference Manual (v3.x). https://github.com/owasp-modsecurity/ModSecurity/wiki/Reference-Manual-(v3.x)
37. Anthropic: Hooks reference (Claude Code). https://code.claude.com/docs/en/hooks
38. Wallace u. a.: The Instruction Hierarchy (2024). https://arxiv.org/abs/2404.13208
39. Model Context Protocol, Specification 2026-07-28: Authorization. https://modelcontextprotocol.io/specification/2026-07-28/basic/authorization
40. Microsoft Learn: Connect your agent to an existing Model Context Protocol (MCP) server (Copilot Studio). https://learn.microsoft.com/en-us/microsoft-copilot-studio/mcp-add-existing-server-to-agent
41. Microsoft Learn: Add tools to custom agents (Copilot Studio). https://learn.microsoft.com/en-us/microsoft-copilot-studio/add-tools-custom-agent
42. Anthropic: Preserved thinking. https://platform.claude.com/docs/en/build-with-claude/preserved-thinking
43. Parecki, Smith: OAuth Client ID Metadata Document, Internet-Draft draft-ietf-oauth-client-id-metadata-document-00 (2025). https://datatracker.ietf.org/doc/html/draft-ietf-oauth-client-id-metadata-document-00
44. Steffen: A simple method for monotonic interpolation in one dimension. Astronomy and Astrophysics 239, 443–450 (1990). https://ui.adsabs.harvard.edu/abs/1990A&A...239..443S/abstract
45. Microsoft Learn: Monitor conversational agent effectiveness (Copilot Studio). https://learn.microsoft.com/en-us/microsoft-copilot-studio/analytics-improve-agent-effectiveness
46. LangChain: Monitor projects with dashboards (LangSmith). https://docs.langchain.com/langsmith/dashboards
47. Langfuse: User Feedback. https://langfuse.com/docs/observability/features/user-feedback
48. OpenTelemetry: Semantic conventions for generative AI metrics. https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-metrics.md
49. Microsoft Learn: Manage agents at scale in Microsoft Foundry Control Plane. https://learn.microsoft.com/en-us/azure/foundry/control-plane/how-to-manage-agents
50. Cline: Systemprompt mit dem Werkzeug `attempt_completion`, Version 3.17.0. https://github.com/cline/cline/blob/v3.17.0/src/core/prompts/system.ts
51. Cline: Rückmeldeschaltflächen an der Abschlussmeldung, Version 3.17.0. https://github.com/cline/cline/blob/v3.17.0/webview-ui/src/components/chat/ChatRow.tsx
52. AutoGPT (Classic): Systemkomponente mit dem Befehl `finish`. https://github.com/Significant-Gravitas/AutoGPT/blob/master/classic/forge/forge/components/system/system.py
53. OpenHands: Pull Request #7002, „[agent] improve finish tool for sonnet 3.7“ (2025). https://github.com/OpenHands/OpenHands/pull/7002
54. Microsoft Learn: Agent Evaluators for Generative AI (Microsoft Foundry). https://learn.microsoft.com/en-us/azure/foundry/concepts/evaluation-evaluators/agent-evaluators
55. AWS: Prompt templates of the built-in evaluators (Amazon Bedrock AgentCore). https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/prompt-templates-builtin.html
56. Yao u. a.: τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains (2024). https://arxiv.org/abs/2406.12045

Kapitel 7

Grenzen

- **Protokollgröße.** Jeder Lauf liest das ganze Protokoll. Kompaktiert werden nur Werkzeugergebnisse; reicht das nicht, endet der Lauf an der Kontextgrenze.
- **Parallelität.** Es gibt höchstens so viele Arbeitsprozesse wie eingestellt, ohne Einstellung zwei. Ein Lauf belegt einen davon für seine ganze Dauer; weitere Aufgaben warten.
- **Zeitlimit.** Jede Aufgabe hat 600 Sekunden PHP-Zeit, dazu kommen die Limits des Webservers. Ein abgebrochener Lauf setzt in einer späteren Runde fort.
- **Feste Laufgrenzen.** 100 Anfragen, 4 MB je Antwort und dreißig Wiederholungen in Folge gelten für alle Konten; nur die Zeitgrenze je Anfrage lässt sich je Konto ändern.
- **Wiederholung nur bei Fehlermeldungen des Anbieters.** Wiederholt wird eine Anfrage nur, wenn die Antwort einen Fehler enthält, der auf das Muster des Templates passt. Kommt keine Antwort zustande oder ist sie kein JSON, endet der Lauf. Während der Wartezeit belegt der Lauf seinen Arbeitsprozess.
- **Abhängigkeit von Anfragen.** Zeitpläne und Hintergrundaufgaben laufen nur, wenn die Plattform Anfragen erhält, durch Benutzer oder durch einen Cron-Job, der eine gewöhnliche Seite abruft.
- **Keine Begrenzung von Ereignisketten.** Löst die Arbeit eines Agenten ein Ereignis mit einem Plattform-Agenten als Empfänger aus, startet jede Auslösung einen neuen Lauf, bei Agenten im Assistentenmodus, sofern der Inhaber nicht anwesend ist. Eine Zähl- oder Zeitgrenze für solche Ketten gibt es nicht.
- **Keine Aktivierung externer Agenten.** Externe Agenten müssen Messenger oder Gedächtnis selbst abfragen.
- **Eine Antwort je Aufruf.** Ketten über mehrere Agenten brauchen ausdrückliche Rückmeldungen mit `call`.
- **Aufgabengrenzen setzt der Agent.** Meldet ein Agent keine erledigte Aufgabe, fasst die Auswertung alle weiteren Läufe des Gesprächs zu einer Aufgabe zusammen, auch über Fehlschläge und Unterbrechungen hinweg. Jede Meldung beginnt eine neue Aufgabe, auch wenn ein Modell sie in einer Schleife wiederholt.
- **Kennzahlen schließen die Umgebung ein.** Wiederholungen nach Überlastung oder Ratenbegrenzung des Anbieters senken die Zuverlässigkeit, fehlgeschlagene Aufrufe eines nicht erreichbaren externen Servers die Tool-Kompetenz, auch wenn das Modell sie nicht verursacht hat.
- **Warten auf eine Antwort.** Endet ein Lauf aus einem Ereignis, einem Aufruf oder einer Benachrichtigung ohne Meldung, weil der Agent auf die Antwort eines Aufrufs wartet, zählt seine Aufgabe in der Zuverlässigkeit 0, bis er sie meldet.
- **Dauer und Tokens je Schritt.** Sie zeigen Tempo und Verbrauch einer Modellanfrage, nicht, wie viele Anfragen ein Agent für eine Aufgabe braucht. Tokens verschiedener Modelle sind nur bedingt vergleichbar, weil die Modelle Text unterschiedlich in Tokens zerlegen. Im Connector-Modus enthält die Antwortzeit die Ausführung der Werkzeuge, sonst nicht; Agenten mit verschiedenem Werkzeugweg sind in der Dauer nicht direkt vergleichbar.
- **Benutzeraufrufe** sind auf 1500 Zeichen begrenzt und haben nur über den Messenger einen Rückweg.
- **Eine Identität je externem Server.** Alle berechtigten Agenten greifen mit derselben Autorisierung zu; beim externen Dienst sind ihre Zugriffe nicht unterscheidbar (6.2).

- **Umfang externer Werkzeuge.** Die Deklarationen aller zugänglichen externen Server gehen in jede Modellanfrage ein; eine Obergrenze für die Zahl der Werkzeuge gibt es nicht. Copilot Studio lässt höchstens 128 Werkzeuge je Agent zu und empfiehlt 25 bis 30 [41]. Der MCP-Server von Linear umfasste bei der Einrichtung auf pwnc.it im Oktober 2026 70 Werkzeuge mit geschätzt rund 24 000 Tokens.
- **Reasoning bei geänderter Werkzeugliste.** Ändert sich die Werkzeugliste eines Gesprächs, verwirft Anthropic das Reasoning der Antworten davor, und der Prompt-Cache beginnt an dieser Stelle neu. Anthropic beschreibt das Verwerfen als Notlösung mit Kosten an Ausgabetokens und nennt Wege, bei denen die Werkzeugliste unverändert bleibt (`defer_loading`, `tool_addition` und `tool_removal`); PWNC nutzt sie nicht [42].
- **Token im Connector-Modus.** Führt der Anbieter die Werkzeuge aus, kann PWNC auf einen 401 eines externen Servers während der Antwort nicht reagieren. Es erneuert Tokens vor jeder Anfrage, wenn sie innerhalb von 60 Sekunden ablaufen.
- **Erweiterte Scopes nur durch neues Verbinden.** Verlangt ein Server zusätzliche Scopes, schlagen die betroffenen Aufrufe fehl, bis ein Verwalter den Server erneut verbindet. Die MCP-Spezifikation empfiehlt Clients, die im Auftrag eines Benutzers handeln, eine Step-up-Autorisierung [39]; PWNC bereitet sie mit der gespeicherten Scope-Vereinigung vor, auslösen muss sie der Verwalter.

Kapitel 8

Rahmenbedingungen und Abgrenzung

- PHP, ein Webserver und MySQL oder MariaDB. Gedächtnis und Klassifikator nutzen die Datenbank; bei MySQL ab 5.7.6 verwendet der Volltextindex den N-Gramm-Parser.
- Hintergrundaufgaben und Zeitpläne werden durch Anfragen an die Plattform angestoßen; ohne Verkehr übernimmt das ein Cron-Job, der eine gewöhnliche Seite abruft.
- Die Installation muss sich selbst per HTTP erreichen können, für den Arbeitsprozess und den eingebauten MCP-Client.
- Plattform-Agenten brauchen Zugang zu einem KI-Anbieter. Führt der Anbieter die Werkzeuge aus (Anthropic), muss PWNC für ihn aus dem Internet erreichbar sein.
- Externe MCP-Clients und externe MCP-Server müssen die Version 2026-07-28 sprechen. Clients älterer Versionen brauchen eine Brücke außerhalb von PWNC, Server älterer Versionen sind nur über die Brücke `mcp_legacy.php` nutzbar (9). Ohne MCP bleibt der HTTP-Zugang mit Schlüssel.
- Externe MCP-Server müssen von der Installation aus erreichbar sein, im Connector-Modus vom Anbieter aus.
- Für OAuth bei externen Servern muss der Autorisierungsserver die Rückleitung an `pwnc/module/mcp.php?mcp_message=callback` annehmen, und der Browser des Verwalters muss sie erreichen. Manche Autorisierungsserver lehnen Rückleitungen über http ab; Linear etwa lehnte bei der dynamischen Registrierung eine http-Rückleitung an eine lokale Domain ab. Ein Client-ID-Metadatendokument verwendet PWNC nur bei einer Installation unter https.

Abgrenzung. Keine Werkzeuge für einzelne Anwendungsfunktionen. Keine Orchestrierungskomponente; Abläufe entstehen aus Aufrufen, Ereignissen und dem Gedächtnis. Keine Aktivierung externer Agenten. Keine Unterstützung älterer MCP-Versionen, weder als Server noch als Client. Externe

MCP-Server nur für Plattform-Agenten, mit einer Autorisierung je Server, nicht je Benutzer. Keine Gesamtnote der Agenten und kein Bezug auf andere Agenten in der Auswertung.

Kapitel 9

Offene Punkte

- **Tokenbegrenzung:** Eine Begrenzung des Tokenverbrauchs je Lauf ist vorgesehen, vor allem für Läufe aus Ereignissen und Aufrufen. Ort und Form der Konfiguration sind offen.
- **Aufbewahrungsfrist der Auswertung:** Die Frist bestimmt auch den Zeitraum der Ansicht (4.12); eine Einstellung in der Oberfläche gibt es noch nicht.
- **Laufgrenzen je Konto:** Die Zeitgrenze je Anfrage ist im Optionstext einstellbar (`timeout`); Anfragezahl und Wiederholungen sind fest. Ob sie je Konto einstellbar sein sollen, ist offen.
- **Adresse des Client-ID-Metadatendokuments:** Sie enthält eine Abfrage (`?mcp_message=client`); nach dem Entwurf soll eine Client-ID keine Abfrage enthalten (SHOULD NOT) [43]. Eine Adresse ohne Abfrage ist zurückgestellt.
- **Ältere MCP-Versionen bei externen Servern:** PWNC selbst spricht nur 2026-07-28. Eine Brücke liegt als eigenständiges Skript `mcp_legacy.php` neben dem Verzeichnis `pwnc/` im Release; sie ist kein Teil des PWNC-Kerns. Sie wird als externer Server mit der Adresse des älteren Servers als Parameter eingetragen (`mcp_legacy.php?url=…`), übersetzt die Anfragen von PWNC in das ältere Protokoll mit `initialize` und Sitzung und spiegelt die OAuth-Metadaten und -Endpunkte des Servers, sodass PWNC sie als Ressource und Autorisierungsserver sieht.