

Fallstudie „Schadensmeldung“ — ein KI-Agent mit Rückruf

Kleine, offene Fallstudie zum Vorführen und Bearbeiten. Sie zeigt, wie ein Prozess einen **KI-Agenten beauftrag**, der sich erst Minuten später meldet — und was passiert, wenn er das nicht tut. Sie baut auf den externen Diensten auf (Fallstudie „CarBoost mit Agent“: der Agent antwortet dort sofort) und ist der nächste Schritt: **der Agent antwortet später, über einen Rückruf**.

Der Prozess

Eine Versicherungsnehmerin meldet einen Schaden. Die Versicherung beauftragt einen Gutachter-Agenten und wartet. Kommt das Gutachten mit der Empfehlung „regulieren“, zahlt die Versicherung ohne weitere Prüfung. Empfiehlt der Agent eine Prüfung, kann er nicht begutachten oder meldet er sich nicht innerhalb von zwei Minuten, übernimmt die Sachbearbeitung.

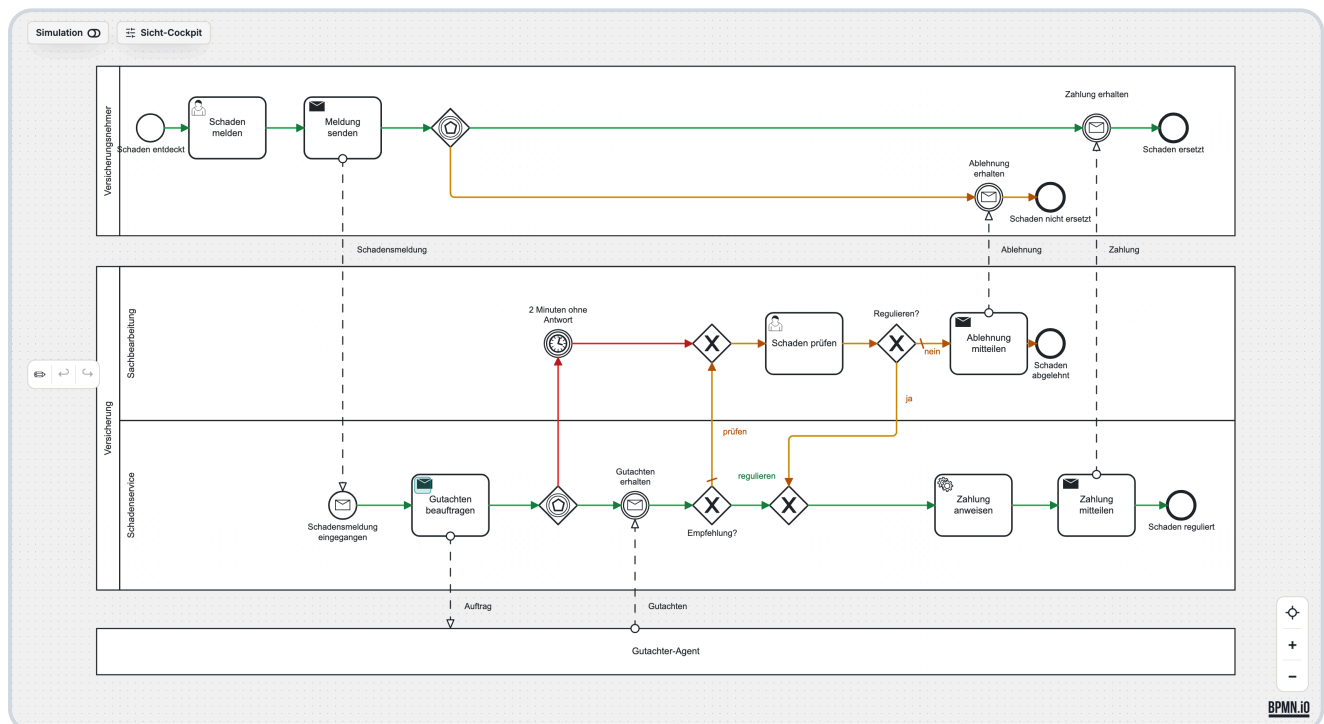


Abbildung 1: Versicherungsnehmer, Versicherung (Sachbearbeitung, Schadensservice) und der Gutachter-Agent als Pool ohne Ablauf

Was das Modell zeigt:

- **Auftrag statt Aufruf.** Der Vertrag liegt an einer *Sendeaufgabe* („Gutachten beauftragen“). Der Prozess wartet nicht an der Aufgabe, sondern läuft zum ereignisbasierten Gateway weiter.
- **Antwort als Nachricht.** Das Gutachten kommt als Nachricht des Agenten am Ereignis „Gutachten erhalten“ an — dieselbe Notation wie zwischen zwei Teilnehmern.

- **Rückfallweg.** Das Zeitereignis „2 Minuten ohne Antwort“ führt zur Sachbearbeitung. Ein Agent, der zu spät antwortet, ändert nichts mehr.
- **Ebenen.** Standardverlauf grün (Gutachten → automatische Zahlung), alternative Verläufe gelb (Prüfung, Ablehnung), Ausnahmeverlauf rot (Zeitlimit).

Der Vertrag

Über das Kontextmenü der Sendeaufgabe: **„Auftrag an einen Agenten...“**. Das Modell nennt nur den Namen der Verbindung (`schaden-agent`); Adresse, Schlüssel und Rückruf-Schlüssel verwaltet die Administration.

Auftrag an einen Agenten — „Gutachten beauftragen“

Das Modell nennt nur den Namen der Verbindung; Adresse und Schlüssel verwaltet der Admin.

Verbindung: Methode:

Echte Aufrufe möglich (Gutachter-Agent (Nachbau)).

Pfad: Zeitlimit (s):

Eingaben — je Zeile: [query|header|path] name = FEEL (ohne Ort: Inhalt)

```
meldung = meldung
betrag = betrag
```

Rückgaben aus dem Rückruf — je Zeile: variable = FEEL über response.body (zustand, ergebnis, grund)

```
empfehlung = if response.body.zustand = "erledigt" then
response.body.ergebnis.empfehlung else "pruefen"
schadensart = if response.body.zustand = "erledigt" then
response.body.ergebnis.schadensart else null
begrueundung = if response.body.zustand = "erledigt" then
response.body.ergebnis.begrueundung else "Kein Gutachten: " +
string(response.body.grund)
```

Die Antwort kommt an: „Gutachten erhalten“. Dort wartet der Prozess (am besten hinter einem ereignisbasierten Gateway mit Zeitlimit).

Probewert — Inhalt des Rückrufs als JSON {zustand, ergebnis, grund}; damit läuft die Simulation ohne Netz

```
{
  "zustand": "erledigt",
  "ergebnis": {
    "schadensart": "Glasbruch",
    "schadenshoehe": 450,
    "empfehlung": "regulieren",
    "begrueundung": "„scheibe“ → Glasbruch; 450 € bis 2000 € – regulieren."
  }
}
```

Abbrechen Entfernen Jetzt testen **Übernehmen**

Gutachter-Agent

Abbildung 2: Vertrag an „Gutachten beauftragen“ — Eingaben, Rückgaben aus dem Rückruf, Probewert

Die Rückgaben lesen `response.body`, den Inhalt des Rückrufs:

```
empfehlung = if response.body.zustand = "erledigt" then response.body.ergebnis.empfehlung else
begrueundung = if response.body.zustand = "erledigt" then response.body.ergebnis.begrueundung
else "Kein Gutachten: " + string(response.body.grund)
```

Vorführen in vier Stufen

1. **Probewert.** Simulation starten, Schaden melden. An „Gutachten beauftragen“ erscheint „Agent arbeitet ...“, der Token wartet am Gateway, nach kurzer Zeit kommt das Gutachten (Probewert: Glasbruch, regulieren) und die Zahlung läuft.
2. **Echter Agent.** In den SIM-Einstellungen „Externe Dienste echt aufrufen“ einschalten. Eine Meldung wie „Wasserrohr im Bad geplatzt, etwa 3.200 Euro“ geht an den Agenten; die Statuszeile meldet „Auftrag angenommen“, wenige Sekunden später kommt der Rückruf (Leitungswasser, über 2.000 € → prüfen) und die Sachbearbeitung sieht die Begründung.
3. **Zwei Fälle gleichzeitig.** Zwei Meldungen kurz hintereinander starten (eine Glasscheibe für 300 €, ein gestohlenes Fahrrad). Jede Versicherungs-Instanz bekommt *ihr* Gutachten — die Rückruf-Marke ordnet die Antwort dem richtigen Auftrag zu.
4. **Zeitlimit.** Eine Meldung mit dem Kursmarker `[langsam]` (der Agent braucht dann fünf Minuten) oder `[still]` (er meldet sich nie). Mit „Keine Antwort (Zeitlimit)“ im Fenster bzw. am Zeitereignis übernimmt die Sachbearbeitung; ein später Rückruf wird verworfen (Statuszeile: „Antwort kam zu spät“).

Weitere Kursmarker: `[fehler]` (Gutachten fehlgeschlagen) und sehr kurze Meldungen (der Agent lehnt ab). Beides führt über die Rückgaben zur Sachbearbeitung.

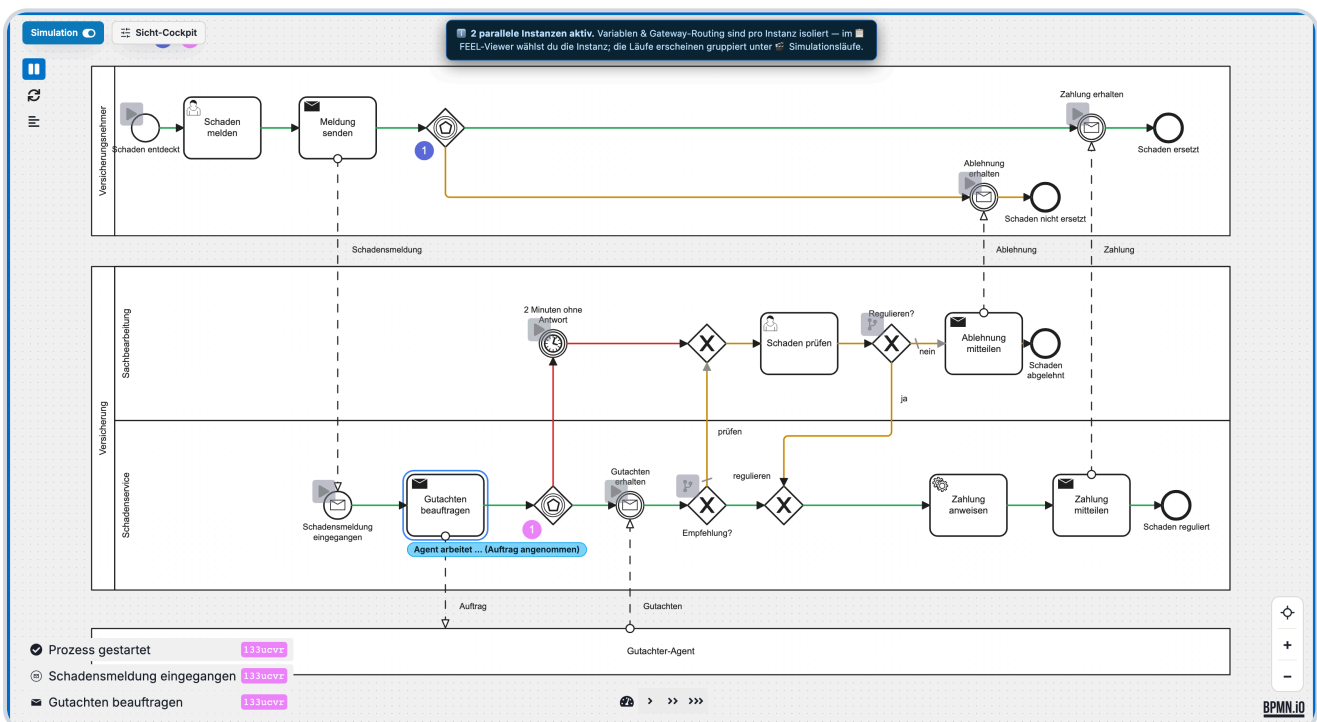


Abbildung 3: Auftrag erteilt — der Hinweis „Agent arbeitet ...“ an der Sendeaufgabe, der Token wartet am Gateway

Einrichtung

Lizenz. Echte Aufträge mit Rückruf brauchen das Paket „Agenten-Rückrufe“ (`agents`, setzt „Externe Dienste“ voraus). Ohne das Paket läuft die Fallstudie mit Probewerten und dem Fenster „Antwort des Agenten“.

Agent starten. Der Schaden-Agent liegt unter `services/schaden-agent` (Node ≥ 20 , ohne Abhängigkeiten; Dockerfile liegt bei):

```
AGENT_KEY=... RUECKRUF_SCHLUESSEL=... node services/schaden-agent/server.mjs # Port 3593
```

UMGEBUNG	BEDEUTUNG
AGENT_KEY	Synixx muss die Kopfzeile <code>X-Key</code> mitschicken (sonst 401)
RUECKRUF_SCHLUESSEL	Schlüssel, mit dem der Agent seinen Rückruf signiert
DAUER_MS	Bearbeitungszeit bis zum Gutachten (Vorgabe 8 s)
LANGSAM_MS	Bearbeitungszeit bei <code>[langsam]</code> (Vorgabe 5 Minuten)
A2A_PUSH	nur A2A: <code>task</code> (Vorgabe, ganze Aufgabe mit Ergebnis) oder <code>status</code> (nur Statusmeldung — Synixx holt das Ergebnis mit <code>GetTask</code>)

Verbindung in Synixx (⚙ → Verwaltung → Verbindungen...): Name `schaden-agent`, Basisadresse des Agenten, Pfad `/auftrag`, Methode POST, Schlüssel-Art „Kopfzeile“ `X-Key` mit demselben Schlüssel, „In der Simulation: echt“. Im Abschnitt **„Aufträge an Agenten (Rückruf)“**: Protokoll „Einfach“, Rückruf-Schlüssel wie `RUECKRUF_SCHLUESSEL`, Gültigkeit in Minuten. Für A2A: Protokoll „A2A 1.0“, erlaubter Pfad zusätzlich `/a2a`, im Modell der Pfad `/a2a` (der Rückruf-Schlüssel gilt bei A2A nicht; dort schützt ein Token je Auftrag). Ein Agent im eigenen Netz braucht „Internes Netz erlauben“.

Rückruf-Adresse. Synixx schickt mit jedem Auftrag die Adresse `<Synixx>/api/agent-rueckruf/<Marke>` mit. Der Agent muss sie erreichen können. Hinter einem Proxy leitet Synixx die öffentliche Adresse aus den Kopfzeilen ab; fest einstellen lässt sie sich mit `AGENT_CALLBACK_BASE` (z. B. `https://live.synixx.de`).

Das Protokoll „Einfach“

```
Auftrag  POST <Basisadresse>/auftrag
        { "auftrag": "a-...", "rueckruf": "https://.../api/agent-rueckruf/<Marke>",
          "daten": { "meldung": "...", "betrag": 3200 } }
        → 202 { "auftrag": "a-...", "zustand": "angenommen" }

Rückruf  POST <rueckruf>      Kopfzeile X-Synixx-Signatur: sha256=<HMAC-SHA256 über den Rumpf>
        { "auftrag": "a-...", "zustand": "erledigt",
          "ergebnis": { "schadensart": "Leitungswasser", "schadenshoehe": 3200,
                       "empfehlung": "pruefen", "begruendung": "..." } }
```

Zustände: `in-arbeit` (Zwischenmeldung), `erledigt`, `fehlgeschlagen`, `abgelehnt` (beenden den Auftrag). Antwortet der Agent gleich mit 200 und einem beendenden Zustand, gilt der Auftrag sofort als erledigt.

Schutz des offenen Eingangs. Die Marke ist zufällig (192 Bit), gilt für genau einen Auftrag und verfällt; ein zweiter Rückruf auf eine verbrauchte Marke bekommt 410. Mit Rückruf-Schlüssel wird jeder Rückruf ohne gültige Signatur abgewiesen (401). Der Rumpf ist auf 64 KB begrenzt, je Absender gilt eine Ratenbegrenzung. Protokolliert werden Verbindung, Auftrag, Zustand und Dauer — keine Inhalte.

Ausführung in Camunda 8

Der Export (Datei → Camunda 8) macht aus dem Auftrag zwei Standard-Konnektoren von Camunda:

- **„Gutachten beauftragen“** wird ein REST-Konnektor. Er sendet den Auftrag im Protokoll der Verbindung; als Auftragskennung dient die Variable `vorgang`, als Rückruf-Adresse der Webhook-Eingang der Camunda-Konnektorlaufzeit (`...:8086/inbound/synixx-schaden-agent-...`).
- **„Gutachten erhalten“** wird ein Webhook-Konnektor. Er nimmt nur beendende Zustände an (Zwischenmeldungen bekommen 422), ordnet die Antwort über den Vorgang zu, prüft — wenn die Verbindung einen Rückruf-Schlüssel hat — dieselbe Signatur wie Synixx (`X-Synixx-Signatur: sha256=...` , gemessen: Camunda nimmt dieses Format an, eine falsche Signatur bekommt 401) und schreibt die Rückgaben mit denselben Ausdrücken über `response.body` wie in der Simulation.

Bei A2A sendet der REST-Konnektor `SendMessage`, die Push-Adresse trägt den Vorgang als Parameter, das Ergebnis kommt aus den Artefakten der Push-Meldung. Meldet ein A2A-Agent nur „erledigt“ ohne Artefakte, bleibt das Ergebnis in Camunda leer (Synixx holt es mit `GetTask` nach, Camunda nicht) — der Fall geht dann zur Sachbearbeitung.

Geprüft gegen Camunda 8.9.13 (`scripts/schulung/c8-schadensmeldung.mjs` , Agent und Synixx-Worker auf einem Rechner im selben Netz): Glasscheibe → regulieren → Zahlung; Wasserrohr 3.200 € → prüfen → Ablehnung; Agent meldet sich nie → nach zwei Minuten Sachbearbeitung; dasselbe mit A2A 1.0. In allen Fällen enden beide Teilnehmerprozesse. Der Lauf nutzt eine Verbindung **ohne** Rückruf-Schlüssel; für den Betrieb gehören der Rückruf-Schlüssel als Secret `SCHADEN_AGENT_RUECKRUF` und die Adresse der Konnektorlaufzeit (an der Verbindung oder als Secret `SYNIXX_WEBHOOK_BASE`) in Camunda. Unterschied zur Simulation: Vorgabewerte der Prozessvariablen kennt Camunda nicht — ohne Gutachten bleibt `empfehlung` dort leer. Für Sendeaufgaben ohne Vertrag (Meldung, Zahlung, Ablehnung) braucht Camunda den Synixx-Worker (`services/c8-worker`).

Einordnung: A2A

Das Protokoll „Einfach“ ist an das offene **Agent2Agent-Protokoll (A2A)** angelehnt, das Google 2025 vorgestellt und an die Linux Foundation übergeben hat (Version 1.0 seit März 2026). A2A kennt ebenfalls Aufträge mit Kennung, Zustände (u. a. *submitted, working, completed, failed, rejected*) und Push-Benachrichtigungen an eine Webhook-Adresse mit Token. Synixx nutzt bewusst die einfachere

Form: ein Aufruf, ein Rückruf, wenige Zustände — gut nachvollziehbar und schnell nachzubauen (der Test-Agent kommt ohne Abhängigkeiten aus). Nicht abgebildet sind u. a. Agent Cards, Rückfragen des Agenten (*input-required*) und Abbruch.

Die Wahl des Protokolls liegt an der **Verbindung**, nicht im Modell. Mit dem Protokoll „**A2A 1.0**“ spricht dasselbe Modell mit einem A2A-Agenten; Synixx ergänzt die Angaben, die das einfache Modell nicht kennt:

SYNIXX- VERTRAG	A2A 1.0 (JSON-RPC)
Auftrag mit den Eingaben	<code>SendMessage</code> , Nachricht mit Rolle <code>ROLE_USER</code> und einem <code>data</code> -Teil (die Eingaben), <code>returnImmediately</code>
Rückruf-Adresse	<code>taskPushNotificationConfig</code> mit Adresse, Token je Auftrag (<code>Authorization: Bearer ...</code>)
<code>zustand</code>	<code>TASK_STATE_SUBMITTED</code> angenommen · <code>WORKING</code> in-arbeit · <code>COMPLETED</code> erledigt · <code>FAILED</code> / <code>CANCELED</code> fehlgeschlagen · <code>REJECTED</code> abgelehnt
<code>ergebnis</code>	erster <code>data</code> -Teil der Artefakte (sonst der Text); meldet der Agent nur „erledigt“, holt Synixx die Aufgabe mit <code>GetTask</code>
<code>grund</code>	Text der Statusmeldung

Der Pfad im Vertrag ist dann der A2A-Endpunkt des Agenten (beim Test-Agenten `/a2a`; seine Agent Card liegt unter `/.well-known/agent-card.json`). Nicht unterstützt sind Rückfragen des Agenten (*input-required*, endet als „fehlgeschlagen“ mit Grund), Streaming, die Bindungen REST und gRPC sowie das Auswerten der Agent Card. Geprüft ist der Adapter mit dem Test-Agenten und gegen einen Agenten aus dem offiziellen JavaScript-SDK (`@a2a-js/sdk` 1.3.0; `scripts/services/a2a-sdk-probe.mjs`). Camunda 8.9 bringt eigene A2A-Client-Konnektoren mit (noch Early Access); der Export von Synixx kommt ohne sie aus und nutzt die allgemeinen REST- und Webhook-Konnektoren.

Dateien

DATEI	INHALT
<code>modelle/Schadensmeldung.bpmn</code>	Musterstand (erzeugt von <code>scripts/schulung/schadensmeldung.mjs</code>)
<code>bilder/</code>	Bilder aus dem echten Werkzeug (<code>scripts/schulung/schadensmeldung-bilder.cjs</code>)
<code>services/schaden-agent/</code>	Test-Agent mit Rückruf (im Repo)

Geprüft durch `test/schadensmeldungModel.test.js` (alle Pfade, kein Deadlock, „Modell prüfen“ ohne Fehler) und `test/e2e-schadensmeldung.js` (Probewert, echter Rückruf, zwei Fälle gleichzeitig, Zeitlimit, A2A mit Nachholen per `GetTask`, Paket nicht freigeschaltet).

Der Durchlauf in Camunda 8 ist ein Skript (`scripts/schulung/c8-schadensmeldung.mjs`), kein automatischer Test — er braucht eine Camunda-Instanz.