Narzędzia i MCP: jak agent dotyka świata

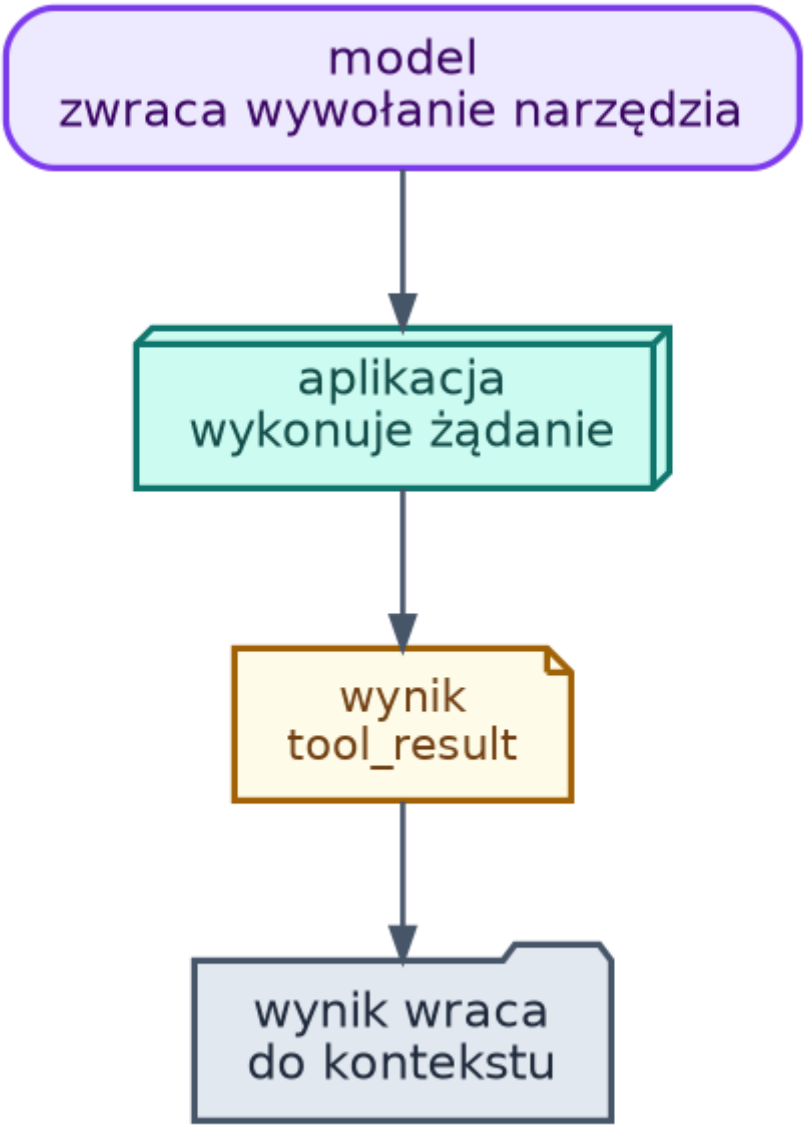
> Strona Różnica pomiędzy czatem a agentem ustaliła, że narzędzie to funkcja z opisem i schematem parametrów. Tu rozkładamy to bliżej: jak wygląda samo wywołanie, co realnie standaryzuje MCP i dlaczego jakość opisu narzędzia decyduje o tym, czy agent w ogóle sięgnie po właściwe.

Skróty użyte na tej stronie (8)

Skrót	Co znaczy
AI	Artificial Intelligence: sztuczna inteligencja, czyli programy naśladujące ludzkie rozumowanie
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
AWS	Amazon Web Services: chmura Amazonu
HTTP	Hypertext Transfer Protocol: protokół stron WWW
IT	Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
MCP	Model Context Protocol: otwarty standard podłączania narzędzi do modeli AI
OAuth	Open Authorization: standard logowania jednej usługi do drugiej bez podawania jej hasła, przez token wydany po zalogowaniu w przeglądarce

Anatomia wywołania narzędzia

> **Pojęcie: wywołanie narzędzia (tool call).** Ustrukturyzowana prośba, którą model zwraca zamiast zwykłego tekstu — nazwa narzędzia plus zestaw parametrów w formacie JSON, zgodny ze schematem, jaki to narzędzie zadeklarowało. Model nie wykonuje niczego sam; zwraca tylko żądanie, a kod wokół niego — nasza aplikacja, nasz agent — faktycznie je wykonuje i zwraca wynik z powrotem do kontekstu jako `tool_result`. Ten cykl żądanie-wykonanie-wynik jest tym samym, co strona o pętli agenta nazywa „myśl, działaj, obserwuj", tylko oglądanym od strony formatu danych, a nie od strony sterowania.

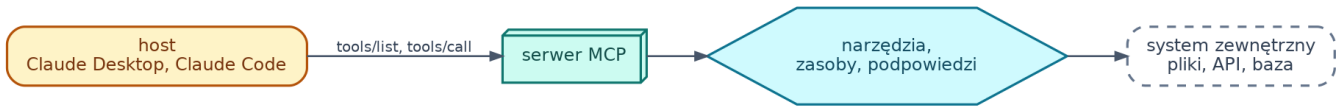


Jeden obrót cyklu: model prosi o narzędzie, aplikacja je wykonuje, wynik wraca do kontekstu (kliknięcie otwiera pełny rozmiar)

Dwa elementy tej anatomii mają bezpośredni wpływ na to, czy agent w ogóle użyje narzędzia poprawnie: opis narzędzia w języku naturalnym, na podstawie którego model decyduje, czy i kiedy po nie sięgnąć, oraz schemat parametrów, który mówi modelowi, jakiego dokładnie kształtu danych narzędzie oczekuje. Oba są częścią promptu systemowego — model widzi je razem z instrukcją, zanim jeszcze dostanie pierwsze pytanie od użytkownika.

Czym jest MCP i co realnie standaryzuje

> **Pojęcie: Model Context Protocol (MCP).** Otwarty standard komunikacji między aplikacją AI (hostem) a zewnętrznymi systemami, opisujący, jak host odkrywa dostępne narzędzia, zasoby i gotowe podpowiedzi wystawiane przez serwer, jak je wywołuje i jak się z tym serwerem uwierzytelnia. Zamiast pisać osobną integrację dla każdej pary „ten model plus ten system zewnętrzny", stawia się jeden serwer MCP przed systemem, a dowolny zgodny ze standardem klient — niezależnie od dostawcy modelu — potrafi się do niego podłączyć.



Host łączy się z serwerem MCP, który wystawia narzędzia do systemu zewnętrznego (kliknięcie otwiera pełny rozmiar)

Konkretnie standard opisuje: sposób odkrywania dostępnych narzędzi (`tools/list`) i ich wywoływania (`tools/call`) w formacie JSON-RPC, dodatkowe prymitywy poza samymi narzędziami — zasoby (`resources`, czyli treści do odczytu, jak pliki czy wpisy w bazie) i gotowe podpowiedzi (`prompts`) — oraz warstwę uwierzytelniania opartą o standard OAuth 2.1.

To, jak szybko się to wszystko zmienia, widać po samej specyfikacji: wydanie z lipca 2026 usunęło dotychczasowy uścisk dłoni `initialize`/`initialized` i identyfikatory sesji na rzecz bezstanowego rdzenia protokołu, dodało routing po nazwie metody i narzędzia wprost w nagłówkach HTTP oraz wskazówki o czasie ważności (`ttlMs`) dla wyników list narzędzi, co pozwala je bezpiecznie cache'ować. Cztery główne zestawy narzędzi programistycznych (TypeScript, Python, Go, C#) obsługują tę wersję, a partnerzy tacy jak AWS, Cloudflare, Google Cloud, Microsoft i Netlify zadeklarowali wsparcie po swojej stronie. Od grudnia 2025 sam protokół nie jest już własnością jednej firmy — Anthropic przekazał jego rozwój nowo powstałej Agentic AI Foundation pod parasolem Linux Foundation, razem z pokrewnymi projektami takimi jak protokół komunikacji między agentami A2A. Dla działu IT to sygnał, że MCP przestał być eksperymentem jednego dostawcy i stał się warstwą integracyjną, na której warto budować, ale też że warto raz na kwartał zerknąć na changelog specyfikacji, zamiast zakładać, że to, co wiemy dziś, będzie prawdziwe za pół roku.

Serwer MCP kontra wtyczka

Serwer MCP jest osobnym, samodzielnym procesem albo usługą, mówiącym jednym, publicznym protokołem — dowolny klient zgodny ze standardem (Claude Desktop, Claude Code, edytor kodu, własna aplikacja) potrafi się do niego podłączyć bez dodatkowej pracy integracyjnej po stronie tego serwera. Wtyczka w rozumieniu zamkniętym — na przykład własny mechanizm rozszerzeń jednej konkretnej aplikacji czatowej — działa tylko wewnątrz tej jednej aplikacji i według jej własnego, niepublicznego kontraktu. Zmiana dostawcy modelu albo aplikacji klienckiej oznacza w tym drugim przypadku przepisanie integracji od zera; w przypadku serwera MCP — tylko podłączenie innego klienta do tego samego serwera.

Gdzie się to dziś spotyka: oficjalne repozytorium przykładowych serwerów MCP obejmuje między innymi dostęp do systemu plików, Gita i popularnych komunikatorów zespołowych; edytory kodu i asystenci programistyczni podłączają się do repozytoriów, baz danych i systemów zgłoszeń przez MCP zamiast przez własne integracje; a w środowisku takim jak nasze naturalnym kandydatem na serwer MCP jest cokolwiek, do czego dziś mamy już API — monitoring, system biletowy, kontroler domeny, wiki.

Dobry opis narzędzia i dlaczego złe opisy szkodzą

Opis narzędzia jest promptem skierowanym nie do człowieka, tylko do modelu, i podlega tym samym regułom dobrego pisania co reszta promptu: jasność, konkret, jednoznaczność. Dobry opis mówi trzy rzeczy naraz — co narzędzie robi, kiedy po nie sięgnąć i kiedy po nie nie sięgać, jeśli istnieje podobne narzędzie, z którym łatwo je pomylić. Opis parametrów powinien być równie konkretny jak opis samego narzędzia — nazwa parametru `path` bez dopowiedzenia, czy oczekuje ścieżki względnej czy bezwzględnej, zostawia model zgadującym.

Najczęstsza, dobrze udokumentowana przyczyna, dla której agent sięga po niewłaściwe narzędzie, to nie ograniczenie samego modelu, tylko dwuznaczny albo niepełny opis — dwa narzędzia o zbliżonych nazwach i nakładających się opisach, brak informacji o tym, kiedy jednego używać zamiast drugiego, albo opis pisany z założeniem wiedzy, której model nie ma. Praktyczna konsekwencja: znacząca część pracy nad jakością agenta to redagowanie opisów narzędzi, nie zmiana kodu, który za nimi stoi — dokładnie ta obserwacja, którą strona Różnica pomiędzy czatem a agentem sygnalizuje jednym zdaniem w ciekawostce o opisie narzędzia jako promptu; tu rozwijamy ją do konkretnej checklisty redakcyjnej.

Narzędzia odczytu kontra narzędzia zmieniające stan

Rozróżnienie między narzędziem, które tylko czyta, a narzędziem, które coś zmienia, decyduje o tym, jaki poziom zaufania i jakie uprawnienia mu przyznajemy — do czego wracamy szerzej na stronie Uprawnienia, prompt injection i zasięg rażenia. MCP formalizuje to rozróżnienie jako adnotacje dołączone do definicji narzędzia: `readOnlyHint` oznacza narzędzie, które nie modyfikuje żadnego stanu; `destructiveHint` ostrzega, że wywołanie może być nieodwracalne; `idempotentHint` mówi, czy wielokrotne wywołanie z tymi samymi parametrami da ten sam efekt co jedno wywołanie. To metadane deklarowane przez autora serwera, nie gwarancje wymuszane przez sam protokół — mimo to są pierwszym miejscem, w którym warto szukać odpowiedzi na pytanie, czy to narzędzie wolno wywołać bez potwierdzenia.

Fiszka

◫ PLIK KONFIGURACYJNY

Opis narzędzia w formacie MCP: sprawdzenie statusu usługi

```
{
  "name": "sprawdz_status_uslugi",
  "description": "Zwraca bieżący stan usługi systemd na wskazanym serwerze (active, inactive, failed) wraz z czasem ostatniego restartu. Używać do diagnostyki i triage'u zgłoszeń. Nie używać do uruchamiania ani zatrzymywania usługi — do tego służy osobne narzędzie restartuj_usluge.",
  "inputSchema": {
    "type": "object",
    "properties": {
      "serwer": {
        "type": "string",
        "description": "Nazwa hosta docelowego, np. srv-app01"
      },
      "usluga": {
        "type": "string",
        "description": "Nazwa jednostki systemd, np. nginx.service"
```

```
      }
    },
    "required": ["serwer", "usluga"]
  },
  "annotations": {
    "readOnlyHint": true,
    "destructiveHint": false,
    "idempotentHint": true
  }
}
```

Zwracamy uwagę na trzy rzeczy w tym przykładzie: pole `description` mówi wprost, kiedy narzędzia używać i, co równie ważne, kiedy nie — wskazuje osobne narzędzie do zmiany stanu; pole `required` wymusza podanie obu parametrów, więc model nie zgaduje, na którym serwerze sprawdzić usługę; adnotacja `readOnlyHint: true` jest sygnałem dla warstwy uprawnień, że to narzędzie kwalifikuje się do automatycznego wywołania bez potwierdzenia człowieka.

MCP = Model Context Protocol: otwarty standard podłączania narzędzi do modeli AI

Z życia wzięte

Z ŻYCIA WZIĘTE

Tomek konfiguruje nowego agenta i dodaje mu dwa narzędzia MCP o niemal identycznych opisach: sprawdz_serwer i pobierz_dane_serwera. Przy tym samym pytaniu agent raz sięga po jedno, raz po drugie, a wyniki nie zawsze się zgadzają. Razem z Tomkiem doprecyzowujemy oba opisy, dopisując, kiedy używać którego narzędzia i co dokładnie każde z nich zwraca. Po zmianie agent za każdym razem wybiera to samo, właściwe narzędzie.

Ćwiczenia

Zadania sprawdzające.

1. Mamy dwa narzędzia o opisach „Pobiera informacje o serwerze" i „Zwraca dane serwera". Jaki konkretny problem to powoduje i jak go naprawić?

Odpowiedź

Opisy są na tyle podobne i ogólne, że model nie ma podstaw, żeby wybrać właściwe w konkretnej sytuacji — może sięgnąć po dowolne z nich losowo albo naprzemiennie. Naprawa polega na doprecyzowaniu obu opisów tak, żeby jasno różniły zakres (np. jedno zwraca dane sprzętowe, drugie stan usług) i dopisaniu w każdym, kiedy używać tego, a kiedy tego drugiego.

2. Narzędzie ma adnotację `destructiveHint: true` i `idempotentHint: false`. Co to razem oznacza dla decyzji o tym, czy agent może je wywołać bez potwierdzenia?

Odpowiedź

Nie powinien — wywołanie może być nieodwracalne, a dodatkowo powtórzenie go nie da tego samego efektu co jedno wywołanie, więc nawet przypadkowe podwójne wywołanie (np. przez retry po błędzie sieci) może pogorszyć sytuację zamiast ją naprawić. To kandydat na bramkę z potwierdzeniem człowieka, nie na automatyzację.

Źródła: Model Context Protocol, dokumentacja: https://modelcontextprotocol.io — Specyfikacja MCP 2026-07-28, zmiany: https://blog.modelcontextprotocol.io/posts/2026-07-28/ — mapa drogowa MCP 2026: https://modelcontextprotocol.io/development/roadmap — MCP dołącza do Agentic AI Foundation: https://blog.modelcontextprotocol.io/posts/2025-12-09-mcp-joins-agentic-ai-foundation/ — Writing Effective Tools for AI Agents, Anthropic: https://www.anthropic.com/engineering/writing-tools-for-agents

---

Revision #35
Created 2026-09-25 10:33:17 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:12 UTC by Tomasz Konieczny