# KI-Framework “OPACA”: Offene, Sprach- und Plattformunabhängige API für Containerisierte Agenten

## Anforderungen, Umsetzung und Werkzeuge

**Projekt:** Offenes Innovationslabor KI zur Förderung gemeinwohlorientierter (Go-KI)

**Autoren:** Tobias Küster, Robert Strehlow, Benjamin Acar, Oskar Kupke, Cedric Braun

**Institution:** GT-ARC gGmbH

**Datum:** 08.07.2025

goKI

Gefördert durch:

Bundesministerium
für Arbeit und Soziales

aufgrund eines Beschlusses
des Deutschen Bundestages

# Inhaltsübersicht

# Einführung und Motivation

Eines der zentralen Ziele des Go-KI-Projekts bestand in der Entwicklung eines Frameworks und daran angebundener Werkzeuge, die bei der Entwicklung von KI-Anwendungen unterstützen sollen. Es existieren bereits zahlreiche Bibliotheken für KI- und ML-Anwendungen für verschiedene Programmiersprachen, sodass der Schwerpunkt für das hier entwickelte Framework darauf gesetzt wurde, diese plattformunabhängig entwickelt, einzusetzen und miteinander verknüpfen zu können. Zu diesem Zweck wurde ein System entwickelt, das Konzepte von Multiagentensystemen mit Microservices und Container-Technologien verbindet und dazu verschiedene Basisfunktionen mitbringt. Das so entstandene System hat somit großes Potential für die einfache Entwicklung verteilter, robuster, skalierbarer, modularer und wiederverwendbarer KI-Anwendungen.

In den letzten Jahrzehnten hat sich das Gebiet der Multiagentensysteme (MAS) erheblich weiterentwickelt, und es wurden zahlreiche Frameworks für deren Implementierung entwickelt. Trotz vieler fortschrittlicher Merkmale und Fähigkeiten sind sie jedoch meist ein akademisches Thema mit begrenzter Anwendung in produktiven Systemen, z.B. in der Industrie, geblieben. Ein mögliches Hindernis ist der Mangel an standardisierten Schnittstellen und Interoperabilität, den typische Multi-Agenten-Frameworks aufweisen. Stattdessen bevorzugt die Industrie in der Regel Microservice-Architekturen, oft in Kombination mit Virtualisierungs- und Container-Technologien, welche die einzelnen Teile des Systems voneinander und von der Ausführungsumgebung entkoppeln.

Die Entwicklung von Microservice-basierten MAS-Frameworks ist bisher relativ begrenzt, obwohl solche Frameworks ein vielversprechendes Gebiet für zukünftige Forschung und Innovation darstellen: Die Vorteile der Skalierbarkeit, Modularität und Fehlertoleranz, die durch Microservices und Containerisierung bereitgestellt werden, haben das Potenzial, die Fähigkeiten und die Leistung von MAS erheblich zu verbessern. Daher haben wir uns entschlossen, ein neues Framework mit dem Namen “OPACA” (Open, Language- and Platform Independent API for Containerized Agents) zu entwickeln, das die Prinzipien einer Microservices-Architektur verkörpert, bei der Agenten in Docker-Containern oder Kubernetes-Pods bereitgestellt werden, während die typischen Eigenschaften von Agenten beibehalten werden, z.B. dass sie autonomer und dynamischer sind als traditionelle Microservices.

Die verschiedenen Microservice-Komponenten sind so konzipiert, dass sie klein, lose gekoppelt und unabhängig voneinander einsetzbar sind. Sie stellen einzelne Gruppen von Agenten dar, während die Containerisierung mit Docker und Kubernetes die Umgebung, in der eine Anwendung läuft, isoliert. Auf diese Weise kann jede Gruppe von Agenten in der MAS autonom arbeiten und Entscheidungen auf der Grundlage eigener Regeln und Ziele treffen. Durch die Nutzung der REST-Schnittstelle können die Agenten miteinander kommunizieren, Informationen austauschen und bei komplexen Aufgaben zusammenarbeiten. Die Interaktionen zwischen diesen Agenten werden durch eine Plattform erleichtert, die als Dirigent fungiert und für die Orchestrierung des Informationsflusses durch das MAS verantwortlich ist.

Die wichtigsten Vorteile der Kombination von Microservices und Containerisierung mit Multiagentensystemen sind:

- Die Skalierbarkeit wird erheblich verbessert, da sich das System durch einfaches Hinzufügen oder Entfernen von Microservices/Agenten bei Bedarf leicht an veränderte Anforderungen anpassen lässt. Dank dieser dynamischen Skalierbarkeit kann das System sowohl kleine als auch große Operationen effizient abwickeln, wodurch eine optimale Ressourcenzuweisung gewährleistet und Engpässe vermieden werden.
- Die Fähigkeit, Fehler zu isolieren, erhöht die Robustheit und Zuverlässigkeit des MAS als Ganzes. DevOps-Praktiken werden durch die Flexibilität der Microservice-Architektur ermöglicht. Aktualisierungen und Erweiterungen können an einzelnen Agenten vorgenommen werden, ohne dass das gesamte System davon betroffen ist.

In den folgenden Abschnitten geben wir einen kurzen Überblick über Hintergründe und verwandte Arbeiten und beschreiben die identifizierten Anforderungen, die an das Framework gestellt werden. Danach wird zunächst die öffentliche Schnittstelle (API) beschrieben, und dann die Referenzimplementierung, die diese Schnittstelle umsetzt. Anschließend werden verschiedene Werkzeuge vorgestellt, die für das Framework erstellt wurden und für alle damit entwickelten Anwendungen zur Verfügung stehen. Zuletzt werden wir das Framework anhand der gesetzten Anforderungen, seiner praktischen Anwendbarkeit und der initialen Projektziele evaluieren.

---

Dieser Bericht basiert in weiten Teilen auf dem Paper: *Acar et al.: "OPACA: Toward an open, language- and platform-independent API for containerized agents,"* erschienen bei IEEE Access, vol. 12, pp. 10 012–10 022, 2024. (https://ieeexplore.ieee.org/document/10398198)

# Hintergrund

Das OPACA-Framework kombiniert Konzepte von Multi-Agenten Systemen mit Microservices und Container-Technologien. Im Folgenden geben wir einen kurzen Überblick über diese Themen.

Der Begriff des Software-Agenten ist nicht ganz klar abgegrenzt, doch im Allgemeinen versteht man darunter ein Softwareprogramm, das in einem verteilten System zusammen mit anderen Agenten agiert und mit diesen kommuniziert – daher der Begriff Multi-Agenten-System – und dabei eine gewisse Autonomie aufweist. Das bedeutet einerseits, dass der Agent eintreffende Anfragen ggf. ablehnen kann, und dass dieser eigenständig seine Umwelt wahrnehmen, auf diese reagieren oder auch proaktiv agieren kann. Über die Jahre wurden verschiedene Multiagenten-Frameworks implementiert, die diese Eigenschaften in variierendem Umfang abdecken, darunter u.a. JADE, JACK, JIAC, JASON, und viele weitere. Zum besten Wissen der Autoren hat jedoch keines dieser Frameworks eine nennenswerte Verbreitung außerhalb der Forschung, was auch mit deren häufigem Mangel an Interaktionsmöglichkeiten mit anderen Frameworks liegen kann.

Microservices sind eine Form der Service-orientierten Architektur, bei der Systeme aus zahlreichen unabhängigen und über standardisierte Schnittstellen kommunizierenden Webservices aufgebaut sind. Dies erlaubt es, die einzelnen Komponenten unabhängig voneinander (und in unterschiedlichen Programmiersprachen, mit anderen Frameworks, etc.) zu entwickeln, unabhängig voneinander zu testen, etc. Container-Technologien, allen voran Docker und Kubernetes, erlauben es, solche Microservices (und andere Anwendungen) in einer abgeschlossenen, vom Host-System getrennten

Umgebung auszuführen. Dies erleichtert es erheblich, komplexe, aus mehreren Microservices aufgebaute Systeme aufzubauen, auszuführen und zu verwalten.

Multiagenten-Systeme und Microservices haben einige Aspekte gemeinsam, in anderen Punkten sind sie einander gegensätzlich, können sich aber zugleich auch gut ergänzen. Dementsprechend gibt es bereits Ansätze, die Multi-Agenten-Systeme mit Microservices und/oder Container-Technologien verbinden, jedoch zum besten Wissen der Autoren noch nicht in der Weise wie bei OPACA.

## Verwandte Arbeiten

An dieser Stelle soll nur ein kurzer Überblick über verwandte Arbeiten gegeben werden. Eine umfangreichere Beschreibung kann dem Paper (Acar et al., 2024) entnommen werden.

Dastani et al. (2015) argumentieren, dass die Industrie verlässliche und vorhersehbare Systeme benötigt, wohingegen Multiagenten-Systeme im Ruf stehen, nicht immer vorhersehbar zu sein. Zudem spielen in der Industrie etablierte Standards eine große Rolle, sodass viele dem Wechsel zu neuen Technologien und Paradigmen prinzipiell kritisch gegenüberstehen. Im Gegensatz dazu haben Microservices den Weg in den Mainstream gefunden. Microservices haben dabei viele Konzepte gemeinsam mit Multiagenten-Systemen (vgl. Collier et al., 2019), in anderen Punkten unterscheiden sie sich aber auch deutlich, etwa durch ihr oft eng umgrenztes Einsatzgebiet und klar definiertes Verhalten. Es gibt verschiedene weitere Arbeiten, die versuchen, Multiagenten-Systeme und Microservices zu kombinieren:

- Hypermedea (Charpenay et al., 2022) ist ein framework für IoT-Agenten, basierend auf JaCaMo (Jason, Moise, CArtAgO), in dem verschiedene Artefakte Funktionalitäten zum System beisteuern können.
- Ein ähnlicher Ansatz kommt von Boissier et al. (2013), die Agenten-, Organisations- und Umgebungs-orientierte Techniken verbinden. Das Framework erscheint jedoch sehr komplex und es ist nicht klar, wie genau es in einem Produktivsystem eingesetzt werden würde.
- SagaMAS (Limón et al., 2018) ist ein Multiagenten-Framework, das Microservices für verteilte Transaktionen nutzt. Hierbei entsprechen die Microservices verschiedenen Agenten und können von anderen Agenten dezentral orchestriert werden.
- Jagutis et al. (2023) nutzen eine Kombination aus Agent-based Modelling (ABM) und Microservices für ein Verkehrssimulations-System, wobei etwa verschiedene Aspekte der Umgebung als Hypermedia-Resourcen dargestellt und als Microservices genutzt werden.

Trotz der Ähnlichkeiten zwischen Microservices und Agentensystemen und der beginnenden Verwendung von REST-Interfaces in Multi-Agenten-Frameworks konnten wir keine Arbeiten finden, die Multiagenten-Systeme, Microservices und Container-Technologien in vergleichbarer Weise miteinander integrieren wie in dem hier vorgestellten OPACA-Ansatz.

# Anforderungen

Das übergeordnete Ziel des Frameworks besteht darin, die Entwicklung verteilter, skalierbarer, wiederverwendbarer und sicherer Anwendungen zu vereinfachen und zu vereinheitlichen. Hierfür wurden die folgenden Anforderungen identifiziert:

- **Offene, standardisierte Schnittstellen:** Die API sollte einfach zu implementieren sein und die Kombination verschiedener Systeme und unterschiedlicher Architekturen ermöglichen, um die Integration von Legacy-Komponenten zu erlauben.
- **Sprachunabhängigkeit:** Es sollte sich um eine generische Schnittstelle handeln, die in verschiedenen Sprachen implementiert werden und auf unterschiedlicher Hardware laufen kann. Verschiedene Programmiersprachen verwenden unterschiedliche Konzepte und haben daher unterschiedliche Stärken. Die API ist sprachunabhängig und ermöglicht es Entwicklern, diese unterschiedlichen Stärken zu nutzen und miteinander zu kombinieren.
- **Modularität und Wiederverwendbarkeit:** Das Framework sollte modular sein und die Wiederverwendbarkeit bestehender Komponenten innerhalb des Systems ermöglichen. Modularität ist ein wichtiges Gestaltungsprinzip für das Softwaredesign und geht oft mit Wiederverwendbarkeit einher.
- **Selbstbeschreibung:** Komponenten des Frameworks sollten selbstbeschreibend sein, sodass sie Informationen über ihre Funktionen und deren Verwendung bereitstellen. Konzepte wie die Web Service Description Language (WSDL) und neuere Ansätze wie OpenAPI und JSON-Schema haben die Vorteile der Bereitstellung einer abstrakten Informationsebene zur Beschreibung der erwarteten Eingabeparameter und Ergebnistypen sowie einer grundlegenden Beschreibung der Absicht der Funktion gezeigt.
- **Multi-Tenancy:** Es soll neuen Akteuren möglich sein, in das System einzutreten, Dienste/Agenten verschiedener Anbieter zu verbinden und daraus neue Werte zu schaffen. Zu den Vorteilen einer solchen Multi-Tenancy gehören Effizienz und Kostenersparnis durch gemeinsam genutzte Ressourcen.
- **Verteilung:** Das System soll die nahtlose Verteilung von Rechenaufgaben ermöglichen, indem es einerseits verschiedene Gerätetypen (etwa Mikrocomputer und Server) miteinander verbindet und andererseits z.B. Replikation ermöglicht wird.

Insbesondere soll nicht die Verwendung einer bestimmten Sprache oder Bibliothek vorgegeben werden. Obwohl die Verwendung einer Agenten-Sprache oder Bibliothek empfohlen wird und die Referenzimplementierung ein solches Framework verwendet, können einzelne Komponenten auch als einfache Webservices implementiert werden oder bestehende Webservices oder Algorithmen umhüllen oder Anfragen an diese weiterleiten, was z.B. die Anbindung von “Legacy” Komponenten oder den Einsatz von Komponenten auf ressourcenbeschränkten Geräten erleichtert.

# Das OPACA Framework und API

Das OPACA-Framework definiert zwei Komponenten: **Agent Container** (AC), die Agenten bereitstellen, welche verschiedene Aktionen anbieten, auf externe Ereignisse reagieren oder proaktives Verhalten zeigen können, und **Runtime Platforms** (RP), die einen oder mehrere Agent Container verwalten und grundlegende Funktionalitäten bereitstellen, z.B. token-basierte Authentifizierung. Die Agent-Container laufen in Docker oder Kubernetes, während die Runtime-Plattform sie untereinander und mit anderen Runtime-Plattformen verbindet.

Die ACs und RPs kommunizieren untereinander und mit der Außenwelt oder externen Tools über eine standardisierte REST-API und stellen Routen bereit, um aktuell ausgeführte Agenten und ihre Aktionen abzufragen, Nachrichten an Agenten zu senden oder Aktionen aufzurufen. Diese Routen

können einfach über HTTP/HTTPS aufgerufen werden; die Parameter und Rückgabewerte werden als JSON-Objekte übergeben. Die folgende Tabelle zeigt einen Überblick über die einzelnen API-Routen, die von den Agenten-Containern (nur oberer Teil) und von der Runtime Platform (oberer und unterer Teil) angeboten werden.

| **HTTP / REST Route** | | **Beschreibung** |
|---|---|---|
| GET | /agents[/{agent}] | Beschreibung eines oder aller Agenten abrufen |
| POST | /send/{agent} | Nachricht an einen Agenten schicken |
| POST | /broadcast/{channel} | Nachricht an Gruppe schicken |
| POST | /invoke/{action}[/{agent}] | Aktion an einem beliebigen oder einem bestimmten Agenten aufrufen |
| GET, POST | /stream/{stream}[/{agent}] | Datenstrom von einem beliebigen oder einem bestimmten Agenten abrufen oder an diesen schicken |
| GET | /info | Informationen der Plattform und aller darauf laufender Container und Agenten abrufen |
| GET | /config | Konfiguration der Plattform abrufen |
| GET | /history | Event-History abrufen |
| POST | /login | Abrufen eines Access Tokens per Nutzername und Passwort |
| GET | /containers[/{container}] | Informationen zu allen oder einem Container |
| POST, PUT | /containers | Neuen Agenten-Container starten, oder bestehenden neu starten |
| POST | /containers/notify | Über Änderungen im Container informieren |
| DELETE | /containers/{container} | Laufenden Container stoppen und entfernen |
| GET | /connections | Verbindungen zu anderen Plattformen anzeigen |
| POST | /connections | Verbindung zu anderer Plattform herstellen |
| POST | /connections/notify | Über Änderung an verbundener Plattform informieren |
| DELETE | /connections | Verbindung zu anderer Plattform entfernen |

## Schnittstelle der Agenten-Container

Die Agenten-Container bieten REST-Routen an, über welche die Runtime Platform und andere Anwendungen Informationen über diese abfragen und mit ihnen interagieren kann:

- Über die “Agents”-Route können Informationen über alle oder einen bestimmten Agenten im Container abgerufen werden.
- Die “Send”- und "Broadcast"-Routen senden eine asynchrone Nachricht an einen Agenten oder eine Gruppe von Agenten.
- Die "Invoke"-Route erlaubt es, eine von einem Agenten angebotene Aktion aufzurufen. Anders als Send ist Invoke synchron und gibt ein Ergebnis zurück. Die verfügbaren Aktionen und deren Parameter und Ergebnistypen können über die “Agents”-Route abgefragt werden.

Jeder Agenten-Container entspricht einer Anwendung und wird von einer Runtime Plattform ausgeführt. Er kann in sich abgeschlossen sein, oder die Funktionen anderer Container auf derselben oder einer verbundenen Runtime Plattform voraussetzen. Neben den obligatorischen REST-Routen

der API kann der Agenten-Container zudem weitere Ports öffnen, über die etwa weitere Funktionen oder eigene Web-UIs angeboten werden können.

## Schnittstelle der Runtime-Plattform

Die Runtime-Plattform bietet verschiedene REST-Routen an, mit denen die auf ihr laufenden Agenten-Container verwaltet werden können. Außerdem kann eine Verbindung zu einer anderen Runtime Plattform hergestellt werden, wodurch alle Agenten-Container auf der Plattform automatisch Zugriff auf die Funktionen der Container auf der anderen Plattform erlangen. Auf diese Weise ist es einfach möglich, komplexe verteilte Systeme aufzusetzen.

- Die “Info”-Route liefert ausführliche Informationen zu der Plattform und allen darin laufenden Containern, deren Agenten und Aktionen zurück.
- Die verschiedenen “Container”-Routen erlauben das Hinzufügen und Entfernen von Containern auf der Plattform.
- Die “Connection”-Routen können analog dazu genutzt werden, um die Plattform mit anderen Plattformen zu verbinden oder die Verbindung wieder zu lösen.

Zusätzlich bietet die Runtime Plattform außerdem alle Routen der Agenten-Container API an, bspw. “Invoke”, “send”, etc. Aufrufe an diese Routen werden automatisch unverändert an die entsprechenden Agenten-Container (oder verbundene Runtime Plattformen) weitergeleitet, auf denen die angefragten Agenten oder Aktionen zu finden sind. Zusätzliche Ports der Container werden direkt veröffentlicht; bei Kollisionen werden diese ggf. auf andere freie Ports gemappt.

Wenn ein Agenten-Container hinzugefügt wird, sucht die Runtime-Plattform das passende Docker-Image und startet einen entsprechenden Container (siehe folgender Abschnitt). Hierbei werden dem Container bestimmte Informationen in Form von Umgebungsvariablen übergeben, darunter dessen eigene ID, Informationen um mit dessen “Parent-Plattform” zu kommunizieren (und darüber mit anderen Containern), sowie gegebenenfalls weitere Parameter dieses spezifischen Containers, die beim Start angegeben werden können.

## Modelle und Klassen

Die folgende Abbildung zeigt einen Überblick über die einzelnen Modellelemente und Klassen, die (in Form von JSON Objekten) in den verschiedenen Routen erwartet oder zurückgegeben werden.

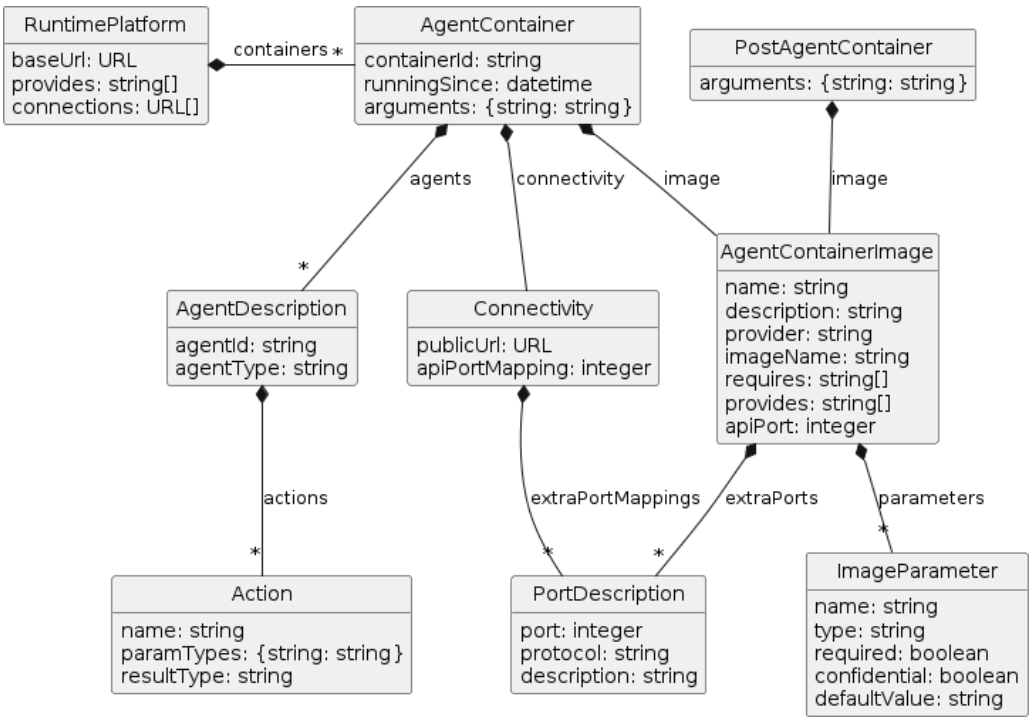


*ABBILDUNG: Modellelemente der OPACA API (Acar et al., 2024)*

# Umsetzung

Im Kern ist OPACA eine API, und keine bestimmte Umsetzung derselben – das Ziel ist es schließlich, Komponenten in verschiedenen Sprachen und Frameworks zusammenzubringen und in beliebigen Umgebungen auszuführen. Im Folgenden werden wir daher lediglich die Referenzimplementierung der OPACA Runtime-Plattform sowie eine Beispiel-Implementierung der Agenten-Container beschreiben. Beide sind als Open-Source-Software auf Github verfügbar.[1] Dort findet sich auch eine ausführliche Dokumentation der Implementierung, der verschiedenen Module, Konfiguration, Umgebungsvariablen, Protokolle, Sicherheitsmechanismen, u.v.m.

## Referenzimplementierung der Runtime Plattform

Die Referenzimplementierung der OPACA Runtime-Plattform wurde in Java mithilfe des Spring Boot Frameworks[2] umgesetzt. Die Plattform selbst ist keine Agenten-Anwendung, sondern im Kern ein Webserver, der die verschiedenen OPACA-Routen anbietet. Das Spring Boot Framework wurde gewählt, weil es die einfache Entwicklung von REST-Servern inklusive Funktionen wie Sicherheit, einfache Web-UIs, Open-API, etc. ermöglicht, sowie aufgrund umfassender Erfahrung mit dem Framework.

In erster Linie implementiert die Runtime Plattform die REST-Routen der OPACA API, wobei viele der Routen lediglich den korrekten Agenten-Container heraussuchen und die Anfrage an diesen weiterleiten, bspw. für den Aufruf einer Aktion eines OPACA-Agenten. Hierbei findet zunächst die Überprüfung der übergebenen Parameter und eine geeignete Fehlerbehandlung statt, inklusive

[1] Quellcode des OPACA-Frameworks: https://github.com/GT-ARC/opaca-core
[2] Spring Boot: https://spring.io/projects/spring-boot

Probieren alternativer Agent-Containers, sollte der erste einen Fehler zurückgeben. Zudem kann zu allen Routen angegeben werden, ob die Anfrage auch an verbundene Plattformen weitergeleitet werden darf, oder nur an einen bestimmten Container, und ob ein Timeout verwendet werden soll.

Neben diesen Routen bietet die Plattform außerdem Routen zum Managen von Agenten-Containern und Verbindungen zu anderen Plattformen an. Agenten-Container können in Docker oder Kubernetes ausgeführt werden, auf demselben Host, auf dem die Plattform läuft, oder auf einem entfernten Server. Die hierfür notwendigen Konfigurationsparameter werden der Plattform beim Start in Form von Umgebungsvariablen übergeben und können danach nicht mehr geändert werden. Soll ein Container gestartet werden, prüft die Plattform zunächst ob das Docker-Image verfügbar ist und startet es dann, wobei wiederum Einstellungen als Umgebungsvariablen übergeben werden – zum einen notwendige Informationen, damit sich der Container mit seiner "Parent-Plattform" verbinden kann, wie dessen URL und gegebenenfalls ein Access Token (s.u.), und zum anderen Parameter, die beim Start vom Nutzer angegeben werden können, bspw. wenn der Agenten-Container seinerseits eine Verbindung zu einem Server oder einer externen Datenbank aufbauen soll.

Wird die Runtime-Plattform beendet, so werden standardmäßig auch alle über die Plattform gestarteten Agenten-Container kontrolliert beendet. Alternativ kann die Plattform auch so konfiguriert werden, dass die zuletzt laufenden Container beim nächsten Start der Plattform in derselben Konfiguration wieder neu gestartet werden, oder auch dass die Container weiterlaufen und sich die Plattform beim nächsten Start wieder mit diesen verbindet – hierbei kann es jedoch sein, dass die Container aufgrund der fehlenden Parent-Plattform nur eingeschränkt funktionieren können. Je nach Anwendungsfall bietet sich die eine oder andere Variante dieses Session-Managements an.

### Docker- und Kubernetes-Integration

Agenten-Container können wahlweise in Docker oder in Kubernetes gestartet werden, wofür verschiedene an die jeweilige Web-API angepasste "Container-Clients" implementiert wurden. Für den Produktiveinsatz wird i.d.R. die Ausführung auf Kubernetes empfohlen, während zum Testen oder wenn Kubernetes nicht verfügbar ist, Docker eingesetzt werden kann. Beide Ansätze funktionieren gleichermaßen gut, aber Kubernetes bietet zusätzliche Funktionen an, etwa das automatische Load-Balancing der gestarteten Agenten-Container in einem Cluster. Agenten-Container sind prinzipiell in beiden Umgebungen ausführbar und können genauso gestartet werden; zusätzlich zu den anderen Parametern können auch Parameter für die "Client-Config" angegeben werden, um so Details anzugeben, die von der Ausführungsumgebung abhängen; hierzu könnte etwa zählen, ob ein Container Zugriff auf GPU-Ressourcen haben soll.

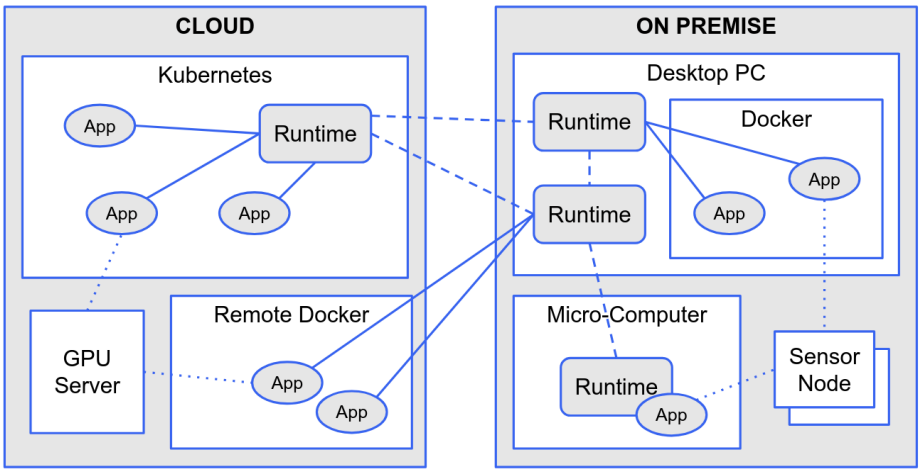


*ABBILDUNG: Beispielhafte Verteilung von OPACA Runtime-Plattforms ("Runtime") und Agent Containers ("App") in einem komplexen und gemischten Deployment. (Acar et al., 2024)*

Wird die Runtime Plattform selbst ebenfalls in Kubernetes deployed, so muss sie über bestimmte Privilegien verfügen, um ihrerseits weitere Pods (Agent Container) starten zu können. Hierfür werden bestimmte Kubernetes *Service Accounts* und *Role Bindings* konfiguriert und alle Komponenten in einem gemeinsamen *Namespace* deployed. Ähnliche Besonderheiten gibt es, wenn die Runtime Plattform selbst in Docker läuft, etwa im Kontext einer umfangreichen Docker-Compose Konfiguration; in diesem Fall muss ihr etwa die URL der Host-Maschine übergeben werden, auf die sie sonst keinen Zugriff hätte. Diese Unterschiede im Verhalten der Plattform werden über Umgebungsvariablen gesteuert, die angeben, in welcher Umgebung die Runtime Plattform selbst und in welcher die Agent Container laufen sollen.

## Basisfunktionen, Validierung und Sicherheit

Neben den Routen der OPACA-API stellt die Runtime Plattform zudem verschiedene Basisfunktionen zur Verfügung, die automatisch von allen darüber gestarteten Agenten-Containern genutzt werden.

- Spring-Boot umfasst eine einfache, aber funktionale Open-API-kompatible Swagger Web UI, über die die verschiedenen Routen der Plattform und der dort laufenden Agenten aufgerufen werden können, einschließlich Validierung der Parameter.
- Eine zusätzliche Schnittstelle für Open-API-kompatible Beschreibungen der Actions der einzelnen Agenten; da diese dynamisch zur Laufzeit hinzukommen können, werden diese nicht direkt von Swagger abgedeckt. Die jeweiligen Parameter- und Rückgabewerte können in Form von JSON-Schema angegeben werden.
- Automatische Validierung der Parameter der aufgerufenen Actions (und Berücksichtigung der Parameter und deren Typen bei der Auswahl des passenden Agenten-Containers), automatischer Failover und Versuch eines alternativen Agenten-Containers bei Fehler, und korrekte Weiterleitung und Behandlung von Fehlern innerhalb der Agenten-Container.
- Alle Routen der Plattform können abgesichert werden. Nach dem Login erhält der Nutzer ein JSON Web Token (JWT)[3], das bei allen folgenden Interaktionen im HTTP-Header übergeben werden kann. Dieselbe Absicherung wird sowohl für Anfragen der Nutzer an die Plattform als

[3] JSON Web Token: https://jwt.io

auch für die Kommunikation zwischen Plattform und Container oder zwischen verbundenen Plattformen verwendet. Eine direkte Kommunikation mit den Containern ist nur mit dem Token möglich, das von der Plattform beim Start für diese generiert wird.[4] Alternativ ist jederzeit auch eine nicht abgesicherte Kommunikation möglich, etwa zum einfachen Testen.

- Darüber hinaus wurde ein einfaches User-Management implementiert,[5] über das zusätzliche Nutzer mit unterschiedlichen Rollen und Rechten angelegt werden können, sodass bspw. nur "Admin"-Nutzer und der Nutzer, der einen Container gestartet hat, berechtigt sind, diesen wieder zu beenden. Die Nutzer werden Session-übergreifend in einer externen Datenbank gespeichert, die zusammen mit der eigentlichen Runtime Plattform gestartet wird.
- Alle Aktivitäten auf der Plattform werden protokolliert, sowohl über das übliche Logging, als auch über eine hierfür eingerichtete Event-History. Diese erlaubt es, verschiedene Events abzurufen und darzustellen (bspw. für externe Plattform-Monitoring-Werkzeuge) sowie sich auf einzelne Event-Typen zu subscriben und sofort eine Benachrichtigung zu erhalten, sobald diese Events ausgelöst wurden, bspw. wenn eine bestimmte Agenten-Aktion aufgerufen wurde.

Darüber hinaus wurde für die OPACA-Runtime-Plattform eine Reihe externer Werkzeuge entwickelt, die automatisch für alle Anwendungen im OPACA-Ökosystem angewendet werden können. Die Werkzeuge werden in einem späteren Teil dieses Berichts näher beschrieben.

## Beispielimplementierung der Agenten-Container

Eine Beispielimplementierung für die Agenten-Container wurde in Kotlin mit dem JIAC VI Framework (Rakow, 2019) umgesetzt. Der allgemeine Aufbau einer JIAC VI Anwendung kann der folgenden Abbildung entnommen werden. JIAC VI erlaubt es, verschiedene "Agenten" auszuführen, die allesamt ihrem eigenen "Life-Cycle" folgen, über eine einfache Notation auf Ereignisse reagieren oder wiederkehrendes Verhalten umsetzen und mit anderen Agenten kommunizieren können.

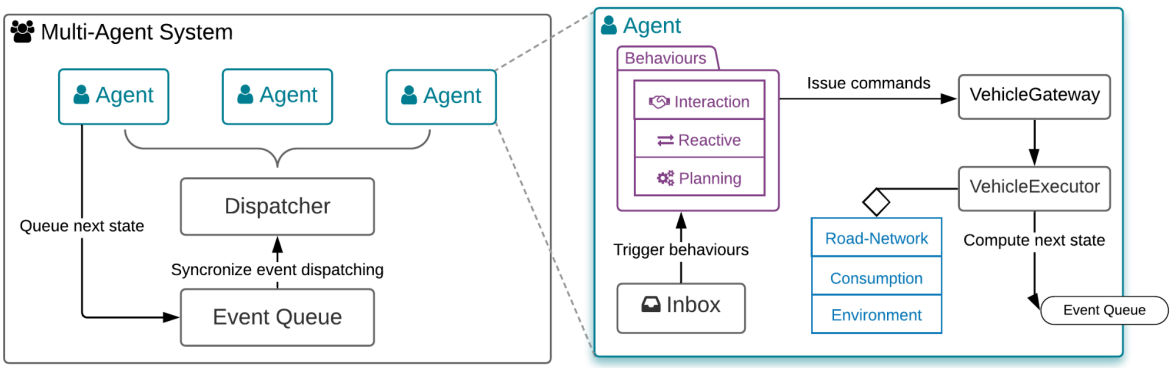


*ABBILDUNG: Generelle Architektur einer JIAC VI Anwendung (Rakow, 2019)*

In jedem JIAC VI Agenten-Container läuft eine Instanz eines "Container-Agent", der als Brücke zur Runtime Plattform und somit zur Außenwelt dient. Er implementiert die OPACA API-Routen des

[4] Aktuell sind nur Access-Tokens mit symmetrischer Verschlüsselung implementiert; eine Umsetzung von asymmetrischer Verschlüsselung ist für die Zukunft geplant.
[5] Für das User-Management stehen weitere REST-Routen zur Verfügung, die nicht in der Tabelle enthalten sind, da diese nicht Teil der OPACA-Kern-API sind.

Agent Containers und leitet bspw. Aktionsaufrufe an den entsprechenden Agenten, der diese Aktion anbietet, weiter; analog für Nachrichten (send und broadcast). Die eigentliche, nach außen sichtbare Logik wird von weiteren Agenten implementiert, die die Superklasse “Abstract Containerized Agent” erweitern. Diese sorgt dafür, dass die Agenten sich und ihre Aktionen automatisch beim Start beim Container-Agent anmelden, und bietet zudem weitere Hilfsfunktionen zum Definieren von Aktionen und zur Interaktion mit dem Container-Agent und der Runtime Plattform. Darüber hinaus können weitere “interne” Agenten definiert werden, die keine eigenen Aktionen nach außen anbieten und mit den anderen Agenten über die von JIAC VI angebotenen Mechanismen interagieren. Für die Interaktion mit der Runtime Plattform wurde ein einfacher Javalin[6] HTTP Server integriert; eine eigene Web-UI oder automatische Parameter-Validierung sind nicht notwendig, da die Routen des Containers nur über die Plattform aufgerufen werden, die diese Funktionen bereitstellt. Gleichsam kann sich die Sicherheit darauf beschränken, ein ggf. vorhandenes Token zu überprüfen, ohne dass Funktionen, selbst Token zu generieren, bereitgestellt werden müssen.

Eine weitere “offizielle” Implementierung der OPACA-API basiert auf Python und FastAPI. Anders als die Implementierung in Kotlin und JIAC handelt es sich hier um einen einfachen Web-Service ohne autonome Agenten, demonstriert damit aber zugleich, wie einfach und vielfältig Agenten-Container implementiert sein können. Das Projekt ist auf PyPI veröffentlicht[7] und die Quellen sind auf GitHub verfügbar. Diese beiden Implementierungen können als Grundlage für Agenten-Container genutzt werden und erlauben dem Entwickler einen einfachen Einstieg in die Entwicklung. Zugleich können Agenten-Container auch in jeder anderen Sprache umgesetzt werden, solange sie die OPACA-API implementieren.

# Werkzeuge

Die einheitlichen Schnittstellen von OPACA ermöglichen es, Werkzeuge bereitzustellen, die automatisch für alle in OPACA entwickelten und auf einer Runtime Plattform laufenden Anwendungen (Agent Container) genutzt werden können. Aktuell stehen die folgenden Werkzeuge zur Verfügung, die jeweils in verschiedenen Abschlussarbeiten umgesetzt und später im Rahmen des Projekts weiterentwickelt wurden:

- Einfache Werkzeuge, die bei der Entwicklung und Administration helfen, etwa ein Command-Line-Interface, eine spezielle Container-Registry, und eine dynamisch generierte Web-UI als Alternative/Ergänzung zu der integrierten Swagger Web UI.
- Ein BPMN-Editor und Interpreter, mit dem OPACA-Anwendungen und ihre Dienste in Form von Geschäftsprozessmodellen verknüpft und diese dann ausgeführt werden können.
- Eine Benutzerschnittstelle, über die der Nutzer per Large Language Model (LLM) auf die verschiedenen Anwendungen zugreifen und deren Dienste ausführen kann.

Sämtliche hier beschriebenen Werkzeuge werden als eigenständige Anwendungen neben der OPACA-Plattform gestartet und verbinden sich mit dieser über die OPACA-API. Zusammengenommen helfen diese Werkzeuge dabei, OPACA-Anwendungen zu entwickeln, zu testen, miteinander zu kombinieren und mit diesen zu interagieren.

[6] Javalin: https://javalin.io
[7] OPACA Python SDK auf PyPI: https://pypi.org/project/opaca

## Command Line Interface

Über Werkzeuge wie “curl” können beliebige REST-Routen aufgerufen werden, und somit auch die von OPACA, aber der Umgang damit ist umständlich und fehleranfällig. Deshalb wurde ein Command-Line Interface (CLI) für OPACA entwickelt, mit dem die verschiedenen API-Funktionen einfacher genutzt werden können (Steinbach, 2024).

Das CLI erlaubt es, wiederkehrende Parameter wie die URL der OPACA-Plattform und die Login-Daten des Nutzers zu speichern und automatisch bei jeder Anfrage zu verwenden. Für die einzelnen Befehle der API (get-containers, invoke, etc.) gibt es eine Auto-Vervollständigung. Die Parameter (der zu startende Container oder Parameter für Services) können aus einer JSON-Datei gelesen oder in einem geführten Formular eingegeben werden. Zudem gibt es zu jeder der Funktionen eine ausführliche integrierte Online-Hilfe.

Das CLI ist für Entwickler gedacht, die ihre OPACA-Anwendungen interaktiv oder automatisch deployen und testen möchten und erlaubt es beispielsweise, die Funktionen in Shell-Scripten mit anderen Command-Line-Werkzeugen zu kombinieren.

## Dynamic Form-Based Web UI

Die integrierte Swagger-UI hilft bei der Nutzung der API, doch für den Aufruf einzelner Aktionen der Agenten mit passenden Parametern gibt es wenig Unterstützung: Der Nutzer muss eigenständig herausfinden, welche Parameter erforderlich sind, und einen passenden JSON-String erstellen. Hier hilft die Dynamische Formular-basierte Web UI. Diese beschränkt sich in ihrer Funktion auf den Aufruf von Aktionen, stellt aber für jede Aktion, gruppiert nach Containern und Agenten, ein automatisch generiertes Formular bereit, über das die nötigen Parameter in strukturierter Form eingegeben und validiert werden können (Seibel, 2024).

Die UI unterstützt bei der Eingabe von Text, Zahlen, booleschen Werte, komplexen und geschachtelten Datentypen (auch rekursiv), und Listen derselben. Die Eingabe komplexer JSON-Strings ist weiterhin möglich, was etwa für lange Listen einfacher ist als über das Formular. Die erwarteten Datentypen und deren Attribute werden aus der Action-Beschreibung und den hinterlegten JSON-Schema-Spezifikationen gelesen und automatisch gegen diese validiert.

Die UI kann von Entwicklern und Nutzern gleichermaßen genutzt werden und erlaubt eine flüssige und fehlerfreie Interaktion mit den verschiedenen auf der Plattform laufenden Agenten und deren Aktionen.

## Agent Container Registry

Die OPACA Agent Container Registry ist ein Verzeichnisdienst, mit dem verfügbare Agent Container registriert, gefunden und direkt auf einer laufenden OPACA-Plattform installiert werden können. Das Verzeichnis stellt Informationen zu den Containern bereit, etwa deren Herausgeber, enthaltene Funktionen, Anforderungen, etc., und erlaubt es, nach bestimmten Attributen zu suchen und zu filtern (Murtaza, 2023). Die eigentlichen Docker-Images sind nicht direkt in der Registry gespeichert, sondern werden von einer gewöhnlichen Container-Registry wie etwa Docker Hub gehostet.

Was die Agent Container Registry von bspw. Docker Hub abhebt ist, dass sie es erlaubt, sich direkt mit einer laufenden OPACA Runtime Plattform zu verbinden (mit den passenden Login-Credentials). Auf diese Weise können die Anwendungen aus der Registry mit einem Klick direkt dort gestartet werden (inklusive eventuell notwendiger Parameter) und laufende Anwendungen können eingesehen und gegebenenfalls wieder gestoppt werden. Somit erleichtert die Agent Container Registry die Administration laufender OPACA Plattformen erheblich.

## BPMN-Editor

Eine weit verbreitete Möglichkeit, bestehende Dienste zu komplexen Anwendungen zu orchestrieren, ist die Verwendung von BPMN-Prozessen (OMG, 2011; Küster et al., 2016), die eine intuitive Notation mit einer umfassenden Ausführungssemantik kombinieren. Sie bieten einen hervorragenden Kompromiss zwischen Benutzerfreundlichkeit, Ausdruckskraft und Selbstdokumentation für viele Anwendungen in der Industrie. Mit der einheitlichen API von OPACA ist es besonders einfach, Aktionen zu durchsuchen und zu kombinieren, die von den auf der Plattform und ihren Containern ausgeführten Agenten bereitgestellt werden. So wurde im Zuge des Go-KI-Projekts ein neuer BPMN-Editor und -Interpreter entwickelt,[8] basierend auf dem BPMN-js Framework[9] (Braun, 2024).

BPMN-js bietet zwar einen hervorragenden webbasierten BPMN-Editor, ermöglicht jedoch nicht die Erstellung ausführbarer Prozesse. Hierzu mussten mithilfe von BPMN *extensionElements* zusätzliche Elemente für z.B. Variablen und Zuweisungen sowie zur Referenzierung von OPACA-Aktionen definiert werden, sowohl im Modell als auch in der Benutzeroberfläche des Editors. Darüber hinaus wurde der Editor um ein Services-Widget erweitert, das eine Verbindung zu einer laufenden OPACA Runtime-Plattform und den Import aller dort verfügbarer Aktionen als Dienste in das BPMN-Diagramm ermöglicht. Auf diese Weise können die OPACA-Aktionen an *Service Tasks* gebunden und zu neuen, komplexen ausführbaren Anwendungen kombiniert werden.

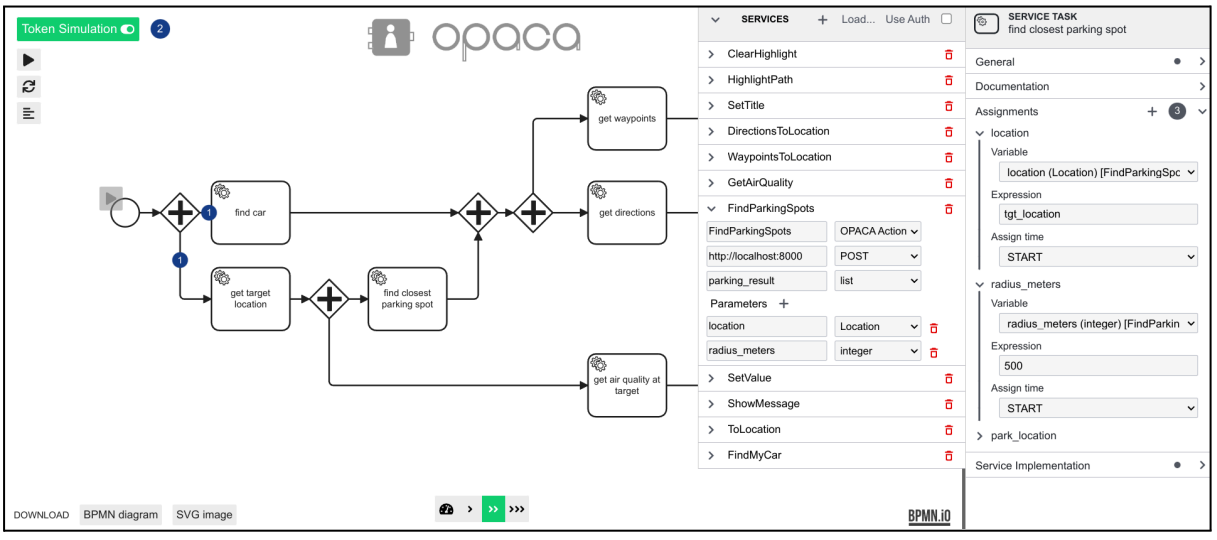


*ABBILDUNG: OPACA BPMN-Editor mit Bearbeitungsbereich (mit aktiver Token-Simulation), Liste der verbundenen Dienste und den Eigenschaften des ausgewählten Elementes.*

[8] Quellcode des OPACA BPMN-Editors: https://github.com/gt-arc/opaca-bpmn
[9] BPMN.io: https://github.com/bpmn-io/bpmn-js

Der Prozessinterpreter verwendet die BPMN-js *Token-Simulation*, die in den Editor selbst integriert ist (und diesen gleichzeitig als Monitoring-Tool nutzt). Wie der Editor musste sie um Unterstützung für Variablen, Zuweisungen und Serviceaufrufe erweitert werden, um Bedingungen automatisch auszuwerten und OPACA-Aktionen aufzurufen. Der Interpreter kann im BPMN-Editor oder während der Ausführung in einem „virtuellen Webbrowser“ innerhalb eines OPACA Agenten-Containers verwendet werden, wodurch auch mehrere BPMN-Diagramme gleichzeitig im „Headless“-Modus geladen und ausgeführt werden können.

Später wurde der BPMN-Editor um eine Integration von ProMoAI[10] erweitert, die es erlaubt, einfache BPMN-Prozesse per LLM generieren zu lassen (Zubenko, 2024).

## LLM-Unterstützung

Darüber hinaus kann für die direkte Interaktion mit einer OPACA-Plattform ebenfalls ein speziell gepromptetes Large Language Model (LLM) genutzt werden. Dieses “kennt” die auf einer OPACA-Plattform laufenden Agenten und deren Aktionen durch eine initiale Abfrage der OpenAPI Spezifikation und kann diese auf Wunsch des Benutzers eigenständig aufrufen; somit ermöglicht es eine dynamischere, intuitivere und niedrigschwelligere Interaktion als der BPMN-Editor.

Die OPACA-LLM-Integration besteht aus zwei Teilen: einem Frontend, implementiert in Javascript mit Vue.js, das eine Web-Benutzeroberfläche bereitstellt, die hauptsächlich aus einem großen „Chat“-Fenster besteht, aber auch Steuerelemente für die Konfiguration der OPACA-Plattform und des zu verwendenden LLM sowie Multimedia-Unterstützung (Sprache und Bilder) enthält, und ein Backend, implementiert in Python, das sich um den Nachrichtenverlauf pro Sitzung kümmert. Dieses verwendet FastAPI, um Anfragen vom Frontend zu akzeptieren, und bietet verschiedene Methoden für die Interaktion mit dem eigentlichen LLM (bspw. LLAMA oder Chat-GPT) und der angeschlossenen OPACA-Plattform[11] (Strehlow, 2024).

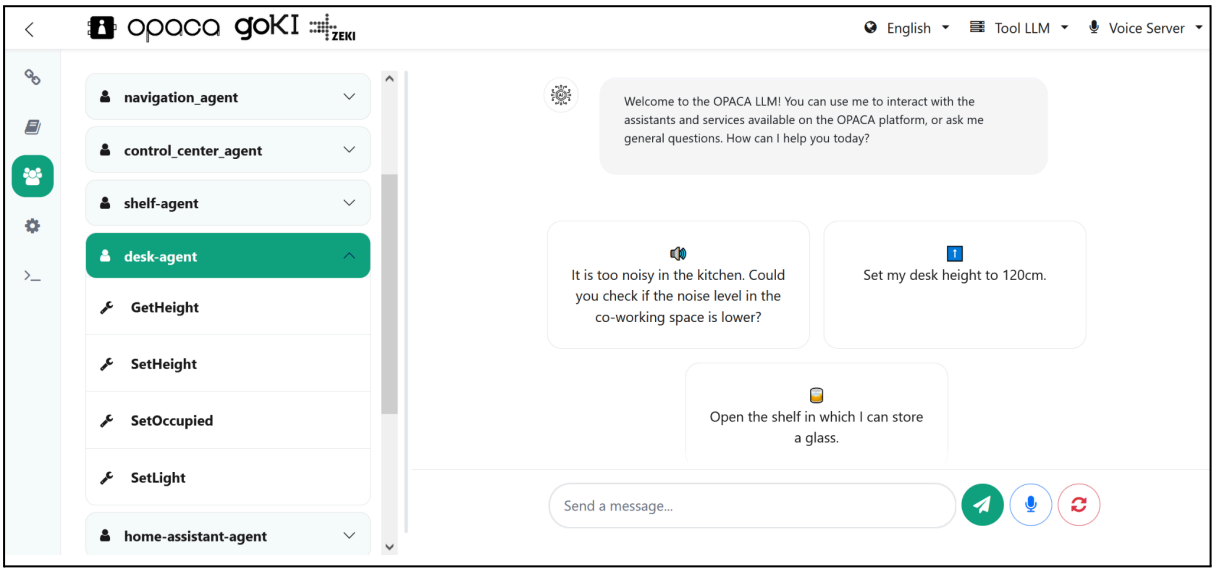


*ABBILDUNG: OPACA LLM-Benutzeroberfläche mit Liste der verfügbaren Agenten und Dienste und großem Chat-Bereich (mit Beispiel-Anfragen).*

[10] ProMoAI: https://promoai.streamlit.app
[11] OPACA LLM-Quellcode: https://github.com/gt-arc/opaca-llm-ui

Für die Interaktion mit dem ausgewählten LLM wurden mehrere Methoden implementiert:

- Ein einfacher Ansatz mit einem einzigen LLM-Agenten,[12] der die verfügbaren Aktionen als Teil des Systemprompts erhält und dessen Antwort nach Aktionsaufrufen durchsucht wird;
- Ein komplexerer Ansatz basierend auf REST-GPT (Song et al., 2023), mit vier LLM-Agenten für die Planung, Auswahl konkreter Aktionen, deren Aufruf und Auswertung der Ergebnisse;[13]
- Ein Ansatz, der zwei LLM-Agenten zum Generieren und Auswerten von Aktionsaufrufen verwendet und die „Tools“-Funktion der meisten (aber nicht aller) neueren LLMs nutzt;
- Ein Ansatz, bei dem für jeden Agenten im OPACA-Framework dynamisch eine Gruppe von LLM-Agenten erstellt wird, die ausschließlich für die Aktionen dieses Agenten verantwortlich ist, sowie weiterer Agenten für die Orchestrierung und Auswahl, welche dieser Agentengruppen verwendet werden sollen, und für die Auswertung der Ergebnisse.

Darüber hinaus kann jeder der Ansätze so konfiguriert werden, dass unterschiedliche LLMs, wie GPT, LLAMA, Mistral und andere verwendet, und übliche Parameter wie die „Temperatur“ des LLM festlegt werden können. Alle Methoden können mehrere Iterationen durchlaufen, bevor sie ein Ergebnis an den Benutzer zurückgeben. Dies geschieht, falls mehrere Aktionen aufgerufen werden müssen oder fehlerhafte Antworten erhalten wurden. Alle Methoden sind auch in der Lage, dem Benutzer mit allgemeinen LLM-Funktionen zu helfen, wie z.B. “Allgemeinwissen”, Textzusammenfassung, usw. – Fähigkeiten, die vom LLM auch verwendet werden, um geeignete Parameter für Aktionen zu generieren, sofern diese nicht explizit angegeben sind, oder um die Ergebnisse auszuwerten und zusammenzufassen, bevor sie dem Benutzer präsentiert werden.

Während das OPACA-LLM im Vergleich zu anderen allgemeineren Ansätzen hinsichtlich der Arten von Diensten, die es ausführen kann, eingeschränkter ist, macht die einheitliche Beschreibung von OPACA-Aktionen die LLM-Integration auch einfacher und robuster: In unseren Tests war das LLM in der Lage, alle verfügbaren Dienste ohne spezielles Training, Finetuning oder umfangreiche Beispiele zu verstehen, korrekt aufzurufen und zu kombinieren, wodurch es sich gut für die hochdynamischen Service-Landschaften eignet, die das OPACA-Framework ermöglichen soll.

# Anwendung und Auswertung

In diesem Abschnitt soll gezeigt werden, wie das OPACA-Framework die anfangs gestellten Anforderungen erfüllt und wie es praktisch eingesetzt werden kann. Das Framework und die Werkzeuge wurden im Go-KI Projekt und darüber hinaus auch in weiteren Projekten aktiv genutzt. Im Folgenden werden insbesondere die Einbettung in das ZEKI-Reallabor sowie weitere Anwendungen beschrieben. Eine Instanz der Plattform läuft rund um die Uhr auf den Servern im ZEKI und kann von Mitarbeitern und Gästen genutzt werden, um mit verschiedenen Funktionen im ZEKI-Reallabor zu interagieren. Die Beobachtungen aus diesen Interaktionen lassen auf eine gute Benutzbarkeit und breite Anwendbarkeit schließen.

[12] Der Begriff „Agent“ kann sich entweder auf einen Agenten im OPACA-Framework oder auf einen LLM-Agenten beziehen, d. h. einen Teil der LLM-Pipeline mit eigenem Zweck, Prompt und Nachrichtenverlauf.
[13] Der REST-GPT-basierte Ansatz wurde in späteren Versionen zugunsten des einfacheren und i.d.R. besser performenden Tool-LLM Ansatzes entfernt.

## Erfüllung der Anforderungen

Im Folgenden werden wir die zu Beginn des Dokuments definierten Anforderungen noch einmal betrachten und beschreiben, wie das vorgestellte Framework zu deren Erfüllung beiträgt.

- **Offene, standardisierte Schnittstellen:** Die Verwendung von REST und HTTP erlaubt die einfache Interaktion und den Datenaustausch mit der Plattform, aus beliebigen anderen Programmiersprachen heraus oder direkt per Command-Line. Docker und Kubernetes sind ebenso weit verbreitet und das Container-Format basiert auf offenen Standards. Microservices erlauben die einfache Integration der Anwendungen untereinander.
- **Sprachunabhängigkeit:** Bibliotheken für HTTP, REST und JSON gibt es für praktisch jede Programmiersprache, und viele andere Funktionen werden von der Plattform bereitgestellt. Durch die Verwendung von Docker spielt die Sprache der Anwendungen somit keine Rolle, und auch die Plattform selbst könnte in einer anderen Sprache nachgebaut werden und sich mit anderen OPACA-Plattformen und den Werkzeugen verbinden.
- **Modularität und Wiederverwendbarkeit:** Die Verwendung von Containern macht die Anwendungen modular; Container können einfach miteinander kombiniert werden, ohne sich gegenseitig zu beeinflussen. Eventuell bestehende Abhängigkeiten zwischen Containern können angegeben und beim Starten berücksichtigt werden. Die Actions können als Microservices einfach von anderen Anwendungen genutzt und kombiniert werden. Die Werkzeuge, wie Registry, BPMN-Editor und LLM-Support machen dies noch einfacher.
- **Selbstbeschreibung:** Die API enthält Routen, die die Plattform als Ganzes, die Container, deren Funktionen, Anforderungen und Agenten, deren Actions, Parameter und Datentypen detailliert beschreiben. Hierbei werden, zusätzlich zu OPACAs eigenen kompakten Formaten, ausgiebig offene Standards wie Open-API und JSON-Schema verwendet.
- **Multi-Tenancy:** Die Runtime Plattform erlaubt es, zur Laufzeit weitere Agenten-Container zu starten oder wieder zu stoppen und sich mit weiteren Plattformen zu verbinden. Auch Agenten-Container können zur Laufzeit neue Agenten und/oder Aktionen anbieten und die Plattform darüber informieren. Die integrierte Benutzerverwaltung sorgt dafür, dass verschiedene Akteure sowie die von ihnen gestarteten Container mit differenzierten Rechten ausgestattet werden können.
- **Verteilung:** Die OPACA-Plattform erlaubt es, Container lokal oder auf einem Remote Docker Host auszuführen; mit der Verwendung von Kubernetes kann die Ausführung der Container auch auf mehrere Nodes verteilt oder repliziert werden. Darüber hinaus können mehrere Plattformen zu einem Verbund zusammengeschaltet werden, der sich nach außen hin weitgehend wie eine einzelne Plattform verhält.

Somit zeigt sich, dass die Anforderungen durch die verwendeten Technologien und die Architektur weitgehend erfüllt oder in ihrer Erfüllung unterstützt werden. Neben der OPACA-REST-API ist es zudem möglich, dass Anwendungen beliebige Ports öffnen (die dann auch auf der Plattform angezeigt werden) und somit weitere Funktionen anbieten können, etwa maßgeschneiderte komplexe Web-UIs oder zusätzliche Interaktionsprotokolle mit externen Anwendungen und Diensten. Zugleich ergeben sich natürlich auch Einschränkungen: So liegt der Fokus etwa klar bei Web-Anwendungen, und die Unterbringung komplexer Anwendungen in einem einzelnen Container kann kompliziert sein. Trotzdem konnte das Framework für die Umsetzung und Integration vielfältiger Anwendungen genutzt werden, wie im folgenden Abschnitt dargelegt wird.

## Integration mit Reallabor-Anwendungen

Für die Integration mit dem ZEKI Reallabor wurden verschiedene Anwendungen als OPACA Agent Containers zur Verfügung gestellt. Für viele Anwendungen, die eine eigene REST-API bereitstellen, z.B. einem Multisensor und einer Smart Kitchen, wurde dazu ein schlanker OPACA-„Proxy"-Agent erstellt, der Aktionen bereitstellt, die intern nur an die eigentlichen Reallabor-Anwendungen delegiert werden, mit zusätzlicher Logik, um sie einfacher verwendbar zu machen (z.B. eine interne Zuordnung von Räumen zu ihren jeweiligen IDs). Weitere Dienste wurden direkt in OPACA entwickelt und bieten zusätzliche Funktionalitäten, die zusammen mit den anderen Reallabor-Anwendungen genutzt werden können.

Die folgenden Anwendungen aus Go-KI und dem Reallabor sind derzeit in OPACA verfügbar:

- Der oben genannte Proxy für mehrere Anwendungen des Reallabors, einschließlich Daten der Multisensoren (Temperatur, Luftfeuchtigkeit, $CO_2$ und Lärm in allen Räumen des ZEKI), Steuerung der Regale in der Küche, Steuerung des Smart Desk und des Wegeleitsystems, sowie der Beleuchtung, Klimageräte und Heizung im ZEKI.
- Ein einfacher Personal Information Manager, der mit der Exchange-API verbunden ist, um Zugriff auf Dienste wie E-Mails, Kalender und Kontakte zu ermöglichen.
- Informationsdienste für z.B. Aktienkurse, Wetter und Navigation unter Verwendung verschiedener kostenloser externer APIs, sowie Dienste, mit denen Google und Wikipedia durchsucht oder eine bestimmte Website geöffnet und ein LLM zum Zusammenfassen der Inhalte verwendet werden können.
- Dienste zum Anzeigen von Text- oder Tabellendaten in einer einfachen Web-UI und zum Generieren verschiedener Arten von Diagrammen für übergebene Daten.
- Generische, auf verschiedene Daten anwendbare ML-Funktionen, beispielsweise für Clustering (DBSCAN) und Dimensionality Reduction (UMAP) von Datenpunkten.
- Ein Proxy für mehrere Funktionen der BeIntelli-API,[14] etwa für die Lokalisierung verschiedener Fahrzeuge in der BeIntelli-Flotte, zum Auslesen von Road-Side-Sensoren und zum Finden freier Parkplätze in der Nähe des ZEKI.
- Ein Proxy-Agent, der Anfragen an zwei LLMs weiterleitet, die auf Fragen rund um die Berliner Verwaltung sowie auf die Zusammenfassung und Visualisierung historischer Daten, z.B. zu Verkehrsunfällen oder der vergangenen Covid-Pandemie, spezialisiert sind.

Ein Test-Deployment des OPACA-System, der Werkzeuge und der oben genannten Anwendungen ist im ZEKI installiert (derzeit nur über das Intranet zugänglich), wo es ohne größere Unterbrechungen seit mehreren Monate für die Interaktion mit der Umgebung bereit steht, sowohl für Entwickler am als auch für Besucher des ZEKI und bei Workshops mit externen Teilnehmern. In diesem Zusammenhang wurden folgende Beobachtungen gemacht:

- Entwickler (aus dem BeIntelli-Projekt, von IOLITE, sowie verschiedene Praktikanten), die mit dem Framework und den Tools vorher nicht vertraut waren, konnten neue Anwendungen schnell und korrekt implementieren und in das bereitgestellte System integrieren.
- Die Werkzeuge, insbesondere die LLM-UI, konnten von Kollegen und Besuchern ohne jegliches Training einfach bedient werden.

[14] BeIntelli-API: https://be-intelli.com/apis-und-sdks/

- Der BPMN-Editor erforderte ein gewisses Verständnis der Notation (und, für ausführbare Prozesse, grundlegende Programmierkenntnisse), war aber ansonsten einfach und intuitiv zu verwenden.
- Die verschiedenen LLMs waren in der Lage, die verschiedenen Dienste im Test-Deployment zu „verstehen“ und mit ihnen zu arbeiten, ohne dass zusätzliches Training, Finetuning oder ausführliche Beispiele erforderlich waren.
- Der BPMN-Editor wurde erfolgreich dazu verwendet, um funktionierende und nützliche Prozesse für das BeIntelli-Projekt zu erstellen, die anderenfalls auf andere Weise hätten implementiert werden müssen.

Auch wenn dies noch keine vollständige und systematische Evaluation darstellt, zeigt es doch deutlich, wie vielfältig die Anwendungen sind, die mit dem KI-Framework und den KI-Tools erstellt werden können, und wie einfach sie auch von ungeschultem Personal genutzt werden können.

Eine qualitative Untersuchung der Benutzbarkeit anhand verschiedener Beispiel-Aufgaben und eines ausführlichen Online-Fragebogens wird zur Zeit durchgeführt, aber bisher liegen noch zu wenige Ergebnisse für eine abschließende Evaluation vor.

## Bezug auf die Projektziele

Das Projektziel war die Entwicklung eines skalierbaren Multiagenten-Frameworks, das verschiedene Kernbestandteile und Infrastruktur für die einfache Entwicklung von KI- und ML-Anwendungen bereitstellt. Hierbei sollte auf eigenen Vorarbeiten aufgebaut und diese hinsichtlich Stabilität und Skalierbarkeit erweitert werden. Das Framework sollte dabei so entwickelt werden, dass die Abläufe nachvollziehbar und transparent sind; zudem sollten Funktionen für generalisierte Künstliche Intelligenz (KI), Machine Learning (ML) und Erklärbare KI (EKI) integriert werden. In weiteren Arbeitspaketen ging es zudem um die Entwicklung von Werkzeugen, u.a. zur Prozessmodellierung, aber auch Werkzeuge für Entwickler, Anwender und zum Testen.

Aufgrund der Vielzahl bereits bestehender Bibliotheken für Machine-Learning-Verfahren, sowie der großen Anzahl verschiedener Programmiersprachen und Paradigmen, in denen diese umgesetzt werden können, wurde die Entscheidung getroffen, den Fokus auf ein plattformunabhängiges System zu setzen, mit dem verschiedene Anwendungen verknüpft und für diese zahlreiche Funktionen und Werkzeuge zur Verfügung gestellt werden können. Das Framework ist hierbei nicht explizit und ausschließlich auf die Entwicklung von KI-Anwendungen ausgerichtet und kann für verschiedene Anwendungsbereiche genutzt werden. Ein Schwerpunkt lag darauf, die Wiederverwendbarkeit und Integration der Anwendungen untereinander zu fördern und zu vereinfachen, wofür entsprechende offene Schnittstellen entwickelt wurden. Durch die Bündelung von Basisfunktionen wie Monitoring, Authentisierung und Benutzerverwaltung in der Plattform wird die Entwicklung der eigentlichen Anwendungen maßgeblich vereinfacht.

Neben der Möglichkeit, sämtliche Abläufe in der Plattform zu protokollieren und durch externe Werkzeuge zu überwachen, wurden ausgewählte KI-, ML- und EKI-Funktionen in Form von Agenten-Containern entwickelt, die auf der Plattform bereitgestellt und von anderen Anwendungen genutzt werden können. Durch die Integration eines Large Language Models mit der Plattform und den darauf laufenden Agenten und Aktionen steht zudem ein breites Spektrum an KI-Fähigkeiten

bereit, etwa für die automatische Auswahl und Ausführung von Diensten und die Zusammenfassung und Interpretation der Ergebnisse.

Neben dieser LLM-Integration über ein Chat-Interface wurden weitere Werkzeuge umgesetzt, insbesondere der im Projektantrag beschriebene Prozesseditor, basierend auf BPMN. Als Grundlage wurde das BPMN-js Framework verwendet, wodurch eine solide Basis und gute Usability sichergestellt werden. Durch die Integration in das KI-Framework können Agenten und Aktionen von dort importiert, zu Prozessmodellen orchestriert, gegen die laufenden Dienste ausgeführt und der Ausführungszustand direkt im Editor nachverfolgt werden. Als Alternative für noch einfachere Orchestrierung wurde Node-RED betrachtet; zudem kann zu einem gewissen Grade auch hier die LLM-Integration genutzt werden. Darüber hinaus wurden weitere Werkzeuge entwickelt, insbesondere zu den im Antrag genannten Bereichen Verzeichnisdienste, sowie Werkzeuge für Entwickler, Nutzer und zum Testen und Monitoring.

## Übertragbarkeit

Um das OPACA-Framework und die OPACA-Tools in einer anderen Umgebung anzuwenden, müssten mehrere der Container für Reallabor-Anwendungen neu geschrieben – oder in manchen Fällen auch nur neu konfiguriert – werden, um der neuen Umgebung gerecht zu werden[15]. Aufgrund der Sprachunabhängigkeit von OPACA und der Nutzung offener und etablierter Standards können neue Agenten-Container jedoch problemlos erstellt werden, indem entweder eine vorhandene API als Proxy verwendet (und mit der umfangreichen Selbstbeschreibung der Agenten und Aktionen von OPACA angereichert) oder der Dienst direkt in einem OPACA-Container in Java, Python, JavaScript oder einer anderen Sprache neu implementiert wird.

Das Framework und die Tools selbst können mit Docker-Compose schnell auf jedem System eingerichtet werden: Alle Komponenten enthalten eine Docker-Konfiguration, mit der sich ein eigenständiger Docker-Container dieser Komponente erstellen lässt, und alles zusammen über eine Docker-Compose Datei konfiguriert und gestartet werden kann. Die LLM-Integration kann verschiedene LLMs nutzen: Je nach Einstellung, Vertraulichkeit und lokal vorhandenen Rechenressourcen kann die GPT-Familie von OpenAI oder eine breite Palette selbstgehosteter Open-Source-Modelle wie LLAMA, Mistral und andere genutzt werden, indem vLLM[16] oder LiteLLM[17] als Proxy verwendet wird.

# Zusammenfassung

In diesem Bericht wurde beschrieben, wie das “KI-Framework” im Projekt Go-KI umgesetzt wurde. Das resultierende Framework “OPACA” kombiniert verschiedene Aspekte und Paradigmen aus den Bereichen der agentenorientierten Softwareentwicklung, serviceorientierten Architektur und Virtualisierung, insbesondere Multiagenten-Systeme, Microservices und Container-Technologien. Das Framework definiert eine API in Form von REST-Webservices, die aus unterschiedlichsten

[15] Siehe hierzu auch den Bericht “Raum-unabhängiges Gesamtkonzept für die Arbeitswelt der Zukunft”, das die Anwendbarkeit und Übertragbarkeit der eigentlichen Reallabor-Anwendungen thematisiert.
[16] vLLM: https://github.com/vllm-project/vllm
[17] LiteLLM: https://www.litellm.ai

Umgebungen und Programmiersprachen genutzt oder selbst angeboten werden kann. Einzelne Anwendungen werden in Form von “Agenten-Containern” angeboten und durch eine “Runtime-Plattform” orchestriert; letztere bietet zudem verschiedene Basisfunktionen an, etwa für Authentisierung, Benutzerverwaltung, Logging und Monitoring, von der alle über die Plattform verbundenen Anwendungen profitieren. Auf diese Weise ist es möglich, vielfältige Anwendungen mit OPACA zu verknüpfen und anwendbar zu machen.

Neben dem eigentlichen Framework wurden verschiedene Werkzeuge umgesetzt, sowohl für die Zielgruppe der Entwickler neuer Anwendungen, als auch für Plattform-Administratoren und Endanwender. Hierzu zählen unter anderem ein Command-Line Interface, ein Verzeichnisdienst, ein BPMN-Editor und ein Chat-Interface zur Interaktion in natürlicher Sprache. Insbesondere diese beiden Werkzeuge, der BPMN-Editor und in noch größerem Maße das Chat-Interface, erlauben eine besonders niedrigschwellige Interaktion mit dem Framework; das Chat-Interface, das ein Large Language Model benutzt, erlaubt zudem die einfache Kombination der OPACA-Dienste mit den generellen KI-Funktionalitäten und Fähigkeiten des LLM.

Das Framework und die Werkzeuge wurden dazu genutzt, verschiedene Anwendungen im Kontext des ZEKI-Reallabor und darüber hinaus umzusetzen, etwa die Integration mit einem Mail-Server, verschiedene Informationsdienste und eine Anbindung an die API des BeIntelli-Projekts. Sie waren über einen langen Zeitraum auf den Servern im ZEKI deployed, wo sie von Mitarbeitern und Besuchern genutzt und ausprobiert werden konnten. Die hierbei gesammelten Erfahrungen, sowohl bei der Entwicklung als auch der Benutzung, waren zum überwiegenden Teil positiv und lassen auf eine breite Anwendbarkeit schließen. Sowohl das eigentliche KI-Framework “OPACA” als auch der BPMN-Editor und das Chat-Interface wurden als Open-Source-Software auf GitHub veröffentlicht. Ein Python-SDK ist auf PyPI verfügbar. Die wesentlichen Konzepte des Frameworks wurden in einem Open-Access-Paper beschrieben und in Workshops vorgestellt. Eine abschließende Evaluation der Akzeptanz und Benutzbarkeit des Frameworks und der Werkzeuge wurde angestoßen, zum aktuellen Zeitpunkt aber noch nicht abgeschlossen.

## Weiterführende Arbeiten

In seiner aktuellen Version kann das OPACA-Framework bereits zuverlässig eingesetzt werden, zugleich gibt es aber auch noch Aspekte, die in Zukunft noch gestärkt werden können. Das OPACA-Framework integriert eine Token-basierte Authentisierung und ein einfaches User-Management, über das die Nutzung der Funktionen eingeschränkt werden kann. Die Kommunikation mit der Plattform kann über HTTPS abgesichert werden. Diese Funktionen haben sich in den bisherigen Tests als funktional erwiesen, sollten aber nochmals von Security-Experten evaluiert und gegebenenfalls erweitert werden, bevor das OPACA-Framework für Anwendungen eingesetzt wird, in denen Sicherheit essentiell ist.

Aktuell ist das KI-Framework im Wesentlichen eine Middleware, die eine API und verschiedene Basisfunktionen anbietet; weitere Funktionen können in Form von Agenten-Containern hinzugefügt werden. Hierzu zählen auch verschiedene generische KI- und ML-Funktionen, die über entsprechende Agenten-Container auf der Plattform installiert und genutzt werden können. Hier wären noch weitere ML-Funktionen denkbar, die im Rahmen des Projekts nicht mehr umgesetzt

werden konnten. Hierzu zählen auch weitere LLM-Funktionen, mit denen im letzten Projektzeitraum experimentiert und die sich als sehr vielversprechend erwiesen haben.

Zuletzt: Das Framework wurde institutsintern in verschiedenen Projekten und für zahlreiche Anwendungen genutzt, doch eine Anwendung in der Breite und durch externe Stakeholder fand bisher nur in eingeschränktem Rahmen statt. Durch die Veröffentlichung auf GitHub und PyPI und ausführliche Dokumentation in Open-Access-Paper und Technischen Reports kann eine weitere Nutzung über das Ende des Projekts hinaus und durch Externe gewährleistet werden.

# Referenzen


1. B. Acar, T. Küster, O. F. Kupke, R. K. Strehlow, M. G. Augusto, F. Sivrikaya, and S. Albayrak, “OPACA: Toward an open, language- and platform-independent API for containerized agents,” IEEE Access, vol. 12, pp. 10 012–10 022, 2024.

2. M. Dastani, ‘‘Programming multi-agent systems,’’ Knowl. Eng. Rev., vol. 30, no. 4, pp. 394–418, 2015.

3. R. W. Collier, E. O’Neill, D. Lillis, and G. O’Hare, ‘‘MAMS: Multi-agent MicroServices,’’ in Proc. Companion World Wide Web Conf., May 2019, pp. 655–662.

4. V. Charpenay, A. Zimmermann, M. Lefrançois, and O. Boissier, ‘‘Hypermedea: A framework for web (of things) agents,’’ in Proc. Companion Web Conf., Apr. 2022, pp. 176–179.

5. O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, ‘‘Multiagent oriented programming with JaCaMo,’’ Sci. Comput. Program., vol. 78, no. 6, pp. 747–761, Jun. 2013.

6. X. Limón, A. Guerra-Hernández, A. J. Sánchez-García, and J. C. Peréz Arriaga, ‘‘SagaMAS: A software framework for distributed transactions in the microservice architecture,’’ in Proc. 6th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT), Oct. 2018, pp. 50–58.

7. M. Jagutis, S. Russell, and R. Collier, ‘‘Simulating traffic with agents, microservices and REST,’’ in Intelligent Distributed Computing XV. Cham, Switzerland: Springer, 2023, pp. 89–99.

8. C. Rakow, “A framework for simulating mobility services in large scale agent-based transportation systems,” in Proceedings of the 2019 Summer Simulation Conference, ser. SummerSim ’19. San Diego, CA, USA: Society for Computer Simulation International, 2019.

9. C. Steinbach, “Developing a command line tool for the FEATS framework according to modern interface standards and usability practices,” Bachelor Thesis, Technische Universität Berlin, 2023.

10. P. Seibel, “Konzeption und Entwicklung einer Benutzeroberfläche für Interaktionen zwischen Komponenten in Multiagentensystemen,” Bachelor Thesis, Technische Universität Berlin, 2023.

11. M. Murtaza, “Design and implementation of a registry service for JIAC++ agent container images,” Bachelor Thesis, Technische Universität Berlin, 2023.

12. Object Management Group (OMG), “Business Process Model and Notation (BPMN), Version 2.0,” Tech. Rep., 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

13. T. Küster, A. Heßler, and S. Albayrak, “Process-oriented modelling, creation, and interpretation of multi-agent systems,” Int. J. Agent-Oriented Software Engineering, vol. 5, no. 2/3, pp. 108–133, 2016.

14. C. Braun, “Web-based BPMN editor for OPACA with capabilities for creating executable BPMN diagrams,” Bachelor Thesis, Technische Universität Berlin, 2024.

15. A. Zubenko, “Design and Development of an LLM Interface for Prompt-based BPMN Process Generation and Visualization,” Bachelor Thesis, Technische Universität Berlin, 2024.

16. R. Strehlow, “Chain-of-Thought Prompting in a Dynamic Multi-Agent System with Large Language Models,” Master Thesis, Technische Universität Berlin, 2024.

17. Y. Song, W. Xiong, D. Zhu, W. Wu, H. Qian, M. Song, H. Huang, C. Li, K. Wang, R. Yao et al., “RestGPT: Connecting large language models with real-world restful APIs,” arXiv preprint arXiv:2306.06624, 2023.