SFTI - project

'Reference Architecture for AI Agent Access to SFTI APIs'

White Paper

Image created with AI (OpenAI's DALL·E)

Authorship: Swiss FinTech Innovations & Acrea
Release: Version 1.0
Date: 26.02.2026

Authors

Acrea

Christof Dornbierer

Marco Seiz

Swiss Fintech Innovations

Stephanie Wickihalder

About SFTI

Swiss Fintech Innovations (SFTI) is an independent association of Swiss financial institutions committed to drive collaboration and digital innovations in the financial services industry. For more information about *Swiss FinTech Innovations*, please refer to http://www.sfti.ch.

About Acrea

Acrea is digital boutique business consultancy based in Zurich with experienced experts in Digital Banking, Digital Pension & Insurance and Digital Health. For more information about *Acrea,* please refer to https://www.acrea.com/.

Content

Management Summary

The rapid evolution of **agentic AI** and **open finance** is fundamentally changing how financial services are accessed, delivered, and consumed. AI agents are evolving from simple assistants into autonomous systems capable of planning, reasoning, and executing complex, multi-step tasks. To unlock their full potential in regulated financial environments, these agents require **secure, standardized, and consent-based access** to high-quality data and functionality.

This white paper examines how **SFTI Common APIs** and open finance standards can be extended to support this emerging interaction model. It positions AI agents not merely as another type of third-party application, but as a **new interaction layer** that will increasingly sit between customers and financial institutions – often embedded in personal devices or operating as part of broader multi-agent ecosystems.

In addition to providing an architectural introduction to AI agents and their interaction with service providers, a central focus of the paper is the **Model Context Protocol (MCP)**, which has emerged as a de-facto standard for structured interactions between AI agents and external services. MCP introduces a standardized way for agents to discover capabilities, access data and tools, and engage in bidirectional interactions. When combined with existing open finance best practices, MCP enables interoperable agent-based access to financial APIs.

Using **pension advice** as a representative example, the paper illustrates how agentic systems can dramatically reduce manual effort and enable more personalized, transparent, and scalable advisory services. At the same time, it highlights that naively exposing existing APIs is insufficient as AI agents might need more context. Instead, service providers must rethink API use cases, provide richer context, and design **capability-oriented interfaces** that guide AI agents safely and efficiently.

The paper also addresses the **security, data protection, and operational implications** of agent-based access. While many risks are extensions of existing API threats, agent-based consumption introduces new challenges related to dynamic behaviour, multi-agent chains, token handling, and data minimization. The paper provides concrete mitigation strategies and operational best practices aligned with regulated financial environments.

Based on this analysis, the paper concludes that:

- AI agent access to financial functionality will become a near-term reality.
- Financial institutions (and their suppliers) should provide **both traditional APIs and MCP-based access** to their services.
- Existing API specifications and consent models need to be extended to support fine-grained, capability-based authorization and discovery.
- MCP reference implementations are essential to validate feasibility and guide standardization.

The paper therefore recommends that **SFTI and its Common API working group** proactively adapt API specifications for agentic use, enrich them with AI-oriented metadata, and develop reference MCP servers. For financial institutions, early engagement with agent-based access models offers a strategic opportunity to shape future client interactions, balance automation with human expertise, and maintain trust in an increasingly AI-driven financial ecosystem.

1. Introduction

1.1 Baseline

Driven by regulatory initiatives like PSD2 in 2015 and the raise of mainstream available large language models in 2022 two trends could be observed:

Trend 1 – AI: With the rapid progress of generative AI, agentic systems are emerging that can independently plan and act. In finance, they unlock new opportunities such as advanced personalized assistants or automated advisory services – provided they have secure access to high-quality data and functionality.

Trend 2 – Open Finance: Initiatives like Common API (SFTI) are laying the technical foundation in Switzerland for standardized and secure data access and new business models.

Examining how these two trends intersect, and particularly how open finance data and functionality can/will be accessed by AI, therefore becomes relevant. It might also define how open finance APIs and SFTI common API in particular and also their corresponding working groups might have to adapt to this new reality.

1.2 Goal of this White Paper

This white paper aims to provide an overview about how open finance APIs will be embedded into the bigger picture of an AI agent driven world. It will touch on the required business capabilities and subsequently the technical architecture, including relevant components and standards needed as well as discuss challenges arising from such integrations and show a possible reference architecture to do so.

Out-of-scope

This white paper will **not** cover the following aspects:

- Establishing guardrails and ethics principles for responsible and secure use of AI in agents and agentic ecosystems.
- Information about how to develop, deploy and operate such AI agent driven applications (development, DevOps and MLOps considerations). See the two SFTI white papers A Scalable Framework for Implementing Artificial Intelligence in Swiss Financial Institutions and Bridging the AI PoC–Production Gap - Keys to Deployment Success in Swiss Financial Industry for some insights into these topics.
- Advise about what particular technologies, tools and libraries should/could be used to implement AI agent driven systems or to expose existing APIs to AI agent applications.

1.3 Approach

This white paper was elaborated in collaboration of a working group (driven by Acrea) and a sounding board consisting of SFTI members and other subject matter experts. Key questions to be addressed where defined by the working group and the sounding board together. Three review rounds combined with two workshops of the working group and sounding board together were conducted.

1.4 Sounding Board Members

- **Adrian Anderegg**, Partner Financial Services, Eraneos
- **Dominik Bartholdi,** ICT Architect, Raiffeisen
- **Sven Biellmann**, Value Stream Lead Open Finance, Oepfelbaum IT Management AG
- **Peter Durrer**, Business Leader, Mastercard
- **Karsten Frey**, Open Finance Lead, Raiffeisen
- **Jürg Gemeinder**, IT-Architecture & Overarching Design, Raiffeisen
- **Thomas Götz**, Deputy Head Enterprise Architecture, PostFinance
- **Holger Harms**, Head Banking Innovation Lab, Swisscom
- **Oliver Hofer**, Senior Enterprise Architect, PostFinance
- **Peter Ivan**, Delivery Partner, Tata Consultancy Services
- **Rahul Karandikar**, Client Partner – Capital Markets & Financial Services, Tata C.
- **Fabian Keller**, Head Trend & Innovation, ZKB
- **Stefan Neumann**, Lecturer Institute of Finance and Law, OST
- **Karthikeyan Ranganathan,** Head of product engineering , Tata Consultancy
- **Jakob Salfeld-Nebgen,** Agentic AI and Innovation, UBS
- **Michael Schläpfer**, Managing Partner & Security Expert, FortIT
- **Chandra Shekaran**, Global Head, Cards & Payments Practice, Tata Consultancy
- **Guilhem Sirven**, Head of Digital Innovation and Trendscouting, BCV
- **Michael Steiner**, Founding Partner & Security Expert, Arxio
- **Stephanie Wickihalder**, President SFTI
- **Pascal Wild**, Head Consulting & Business Development, ti&m

We would like to express our gratitude to the members of the sounding board for their expertise and engagement in the preparation of this white paper.

2. Big Picture

2.1 Positioning of Agent-based Interactions throughout the Financial Ecosystem

2.1.1 Strategic Potential and Emerging Opportunities

Following the release of the Model Context Protocol (MCP) in November 2024 there have been extensive discussions on agent-based access to APIs. MCP and other protocols will be covered later in this white paper.

MCP can be described as a wrapper layer designed to streamline interactions between models and external APIs. However, this characterization underestimates the potential of the new interactions made possible by such standards. MCP represents a foundational shift toward structured and interoperable model-to-service communication, enabling AI systems not merely to “call” services, but to reason across distributed data, actions, and knowledge resources.

When properly implemented, such standards can transform how organizations operationalize both generative and agentic AI capabilities. Rather than serving as a thin access layer, they allow to bind prompts, contexts, tools, workflows, and enterprise knowledge into cohesive agentic loops. This expands use cases well beyond simple query execution into strategically valuable domains such as:

- Prompt orchestration and reuse: Centralized management of prompt templates, and contextual metadata across tasks and domains.
- Resource and data federation: Secure connection of dispersed enterprise APIs, document repositories, and contextual data sources through standardized schema definitions.
- Tool abstraction and execution: Encapsulation of business functionality as composable tools that agents can discover, invoke, and combine.
- Autonomous task management: Integration with planning components that allow agents to decompose goals, select resources, and dynamically coordinate multi-step actions.
- Governance and observability: Standardized logging and telemetry across tool interactions, supporting compliance, traceability, and model risk management.

In such an architecture, the standard acts as an enabler for agentic ecosystems - where AI models, tools, and policies communicate via shared context rather than ad hoc integrations. Such interoperability will be critical as financial, healthcare, and industrial sectors embrace modular AI infrastructures aligned with trust, compliance, and scalability requirements.

2.1.2 Potential of Agentic Systems in Finance

With the rapid progress of generative AI, **agentic systems** are emerging - software agents that can plan and execute tasks autonomously. In financial services, these agents open up new possibilities ranging from **personalized assistants** to **partially or fully automated advisory**. To operate reliably at scale, two preconditions must be met:

- **Secure and consent-based access** to high-quality data via standardized interfaces (e.g., Common API/SFTI), including logging, and clear data-quality rules.
- **A consistent context standard** which defines how applications expose tools, data, and state to agents. This enables stable orchestration across heterogeneous systems.

Agent-based interactions have the potential to transform the **banking and insurance** landscape. The impact is already visible - and will deepen significantly - in two domains:

- **Internal processes** (e.g., onboarding, case triage, document preparation, risk and compliance checks)
- **Customer self-service and advisory** (guided self-service, advisor co-pilots)

Example: Pension Advisory and Holistic Financial Planning

Today, pension and financial planning services are often costly and labour-intense. They require specialized expertise (including the knowledge of local rules and regulations), substantial preparation on the bank's side, and - for clients - the tedious effort of **assembling documents**.

With agentic support, this journey changes fundamentally:

- **Data capture & aggregation:** With client consent, agents automatically collect relevant information - such as the **three-pillar pension data** and details on accounts, policies, assets, and liabilities - via **Common APIs**, replacing manual collection.
- **Preparation & data quality:** An orchestration agent checks completeness, deduplicates, normalizes formats (e.g., contribution histories, benefit projections), and highlights gaps.
- **Scenarios & optimization:** A planning agent simulates **tax and contribution strategies**, retirement gaps, reaction to lifetime events, cash flows, market-assumption sensitivities, and stress scenarios - quantifying trade-offs transparently.
- **Compliance & suitability:** A control agent validates proposals against product governance, local regulation, and suitability criteria; exceptions escalate to a human advisor. Additionally, it also validates the performance, accuracy and precision to ensure continuous model accuracy.
- **Advisor co-pilot & client self-service:** Advisors receive a concise brief (goals, constraints, scenarios, risks), while clients explore "what-if" questions through guided self-service - with clear explainability and recorded consent.

The example outlined above highlights the transformative power of agentic systems which are poised to fundamentally reshape the operational models of diverse financial ecosystem participants. The implications, while broad, touch each sector distinctly.

For ecosystem participants, the emergence of agentic systems is not merely a technological upgrade - but an opportunity to fundamentally reimagine business models and client engagement. Those who act early to integrate secure data interfaces and standardized context protocols set themselves apart - offering scalable, high-quality service and winning trust through transparency and reliability. The most successful institutions will be those that combine the precision and scalability of agents with human expertise, creating hybrid models that drive both efficiency and superior client outcomes, positioning themselves at the client interface.

2.1.3 High-Level Business Architecture

From a business architecture perspective, AI agent access to APIs will most prominently change the way a customer interacts with a service provider. In finance, the **traditional** interaction model was through digital channels (mobile/web) and personal contact. This then emerged into an **open finance** model, where customers can also interact with service providers through trusted third parties (TPA) that are using APIs to access data and functionality. With the rapid progress made in AI (most prominently LLMs) these channels are enhanced with AI capabilities.

However, another shift in the interaction model can already be seen. Customers will start to interact with service providers via their **personal AI agents** (e.g. embedded into the mobile phones operating systems) and also via **specialized AI agents** (e.g. provided by financial service providers themselves and by fintechs). It is also becoming clear that the next stage in the evolution of the interaction model will be the communication of these AI agents with other AI agents, e.g. specialised ones, to form an **agentic ecosystem** that can plan, interact and solve higher-level tasks.

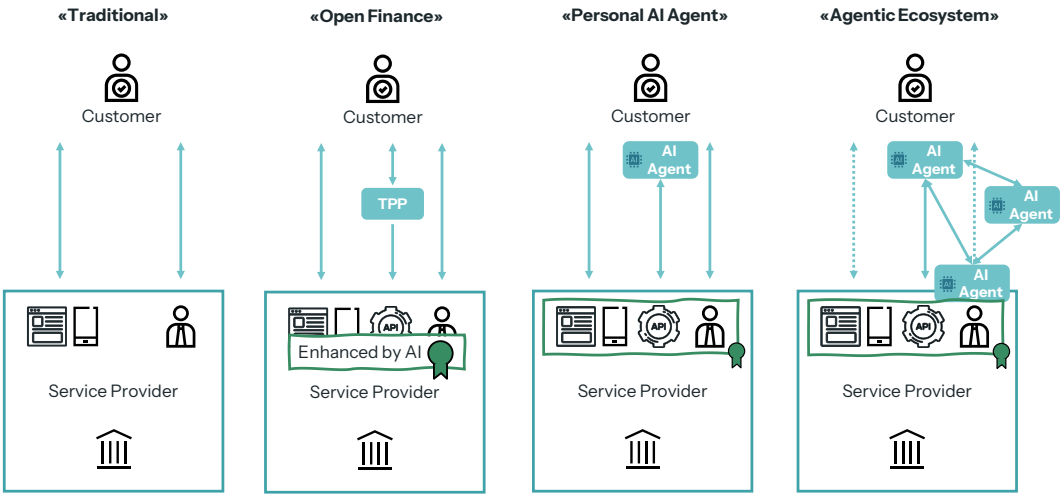

Figure 1: Evolution of the interaction model

From a service providers perspective this access and evolved interaction model mainly affects the **API channel** which will need to be extended to better suit the needs of these agentic use cases. However, the new interaction model might come with new requirements and as (business) APIs base on various internal domains there might be more fundamental functional capabilities needed throughout the organization. In addition, the new interaction model will most certainly affect non-business functional areas like security or compliance.

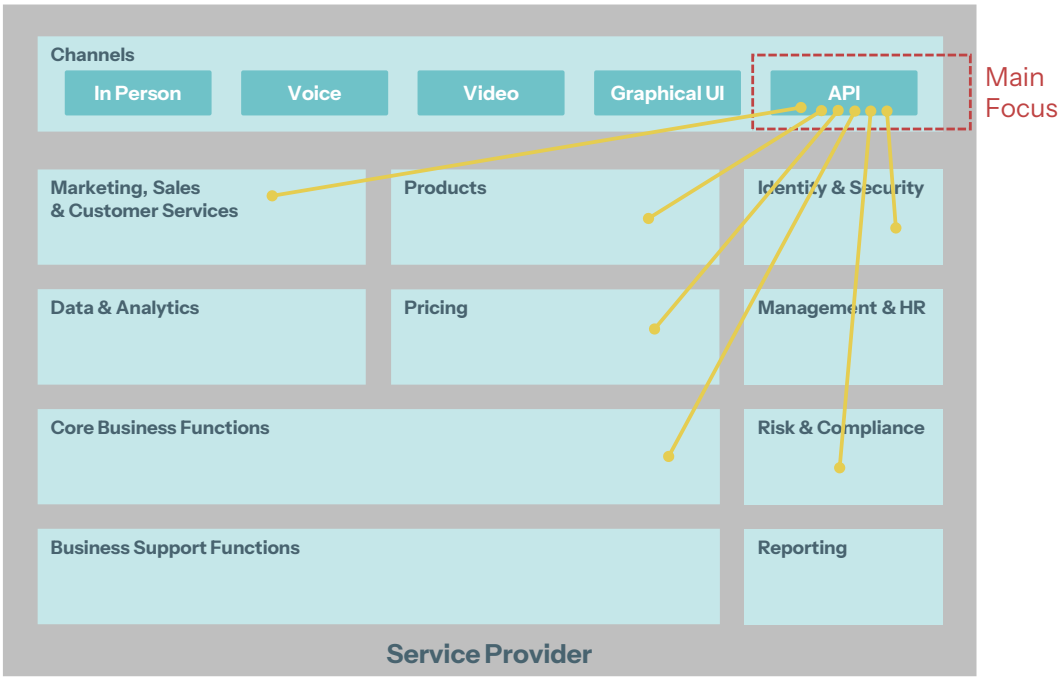

Figure 2: High-level business architecture of a service provider

To understand how different an agentic access to APIs might be we'd like to drill down not only on the service provider side but also the service consumer side. A typical AI agent might need the following capabilities:

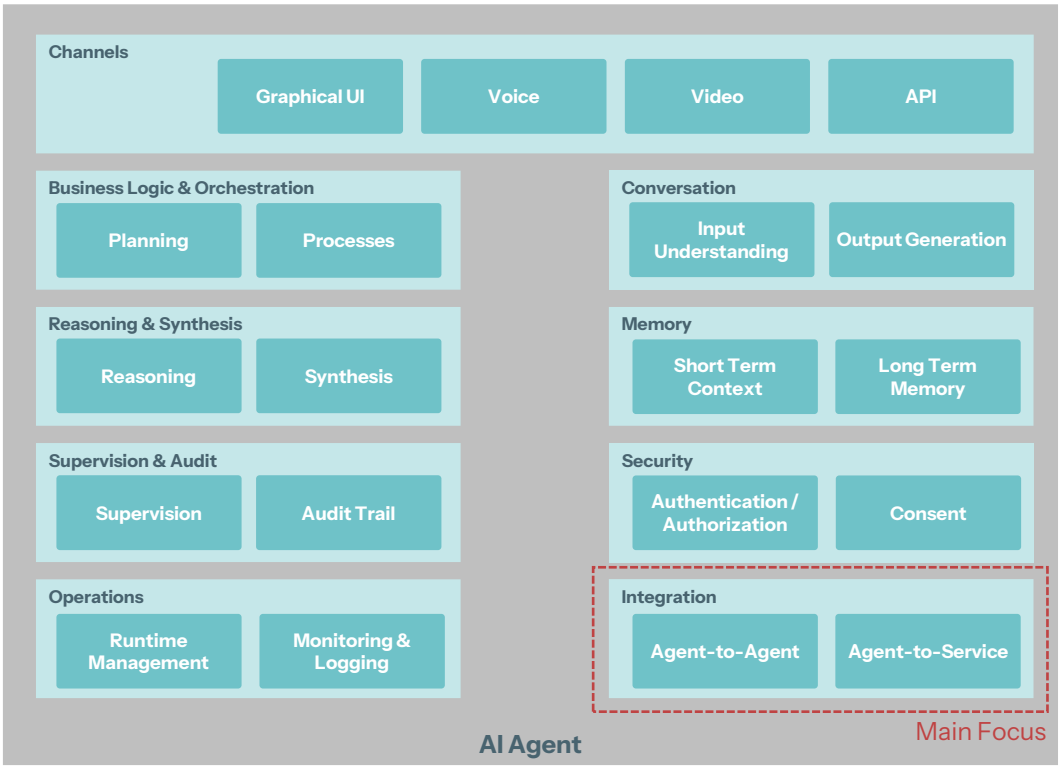

Figure 3: High-level business architecture of an AI agent

Channels: Capabilities used to interact with the agent.

Business Logic & Orchestration: Business domain specific implementation of process and planning capabilities needed to achieve the agents "business goals"

Conversation: Generic functionality to understand input and generate output. Typically implemented by a large language model (LLM)

Reasoning & Synthesis: Capabilities to reason and derive insights from context

Memory: Generic short and long term memory to keep context

Supervision & Audit: Capabilities to supervise, constrain, and intervene in agent behaviour, including enforcement of business and regulatory constraints and to keep track of key events, inputs,… to document the agents actions in the overall process

Security: Handling of customer/system authentication, authorization and consent management

Operation: Capabilities to manage and monitor the runtime of an AI agent, including logging its actions

Integration: Capabilities to integrate with data, functionality and other agents.

2.1.4 High-Level Technical Architecture

An agentic system as described above will most likely look something like the following overview:

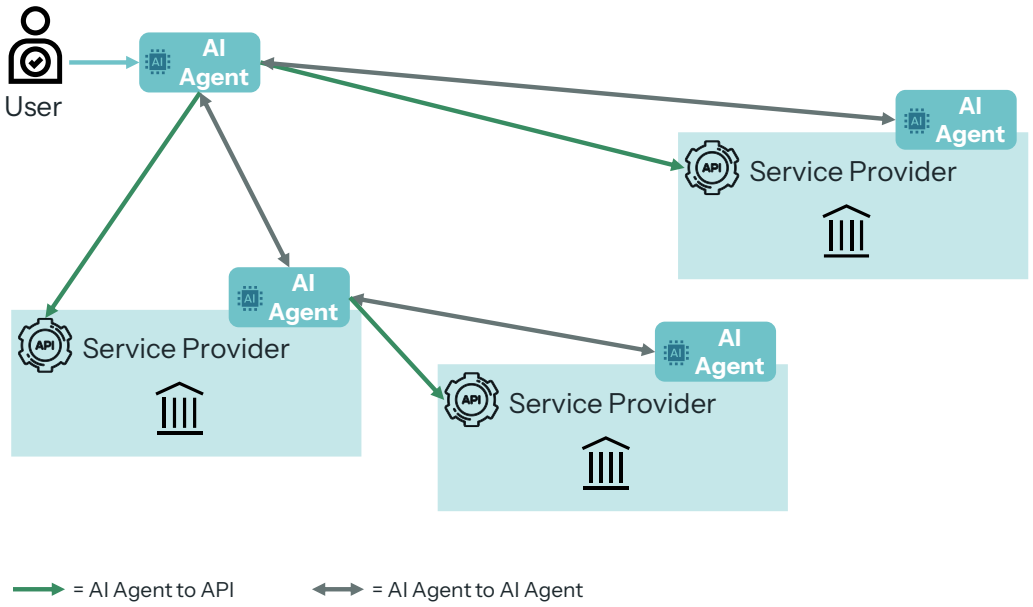

Figure 4: Example of an agentic interaction topology

Users will interact with an AI agent, for example through a chat, voice conversation or even embedded in a more sophisticated application. We're also seeing the rise of personal AI agents like e.g. the OpenAI app or the integration of AI agents into Google and Apple mobile operating systems.

Such an AI agent in term will need access to external systems. We can distinguish between the following main external system types:

Access to data/functionality via APIs: e.g. access to the users bank account data, submit a payment or call a specific simulation API

Conversation with other agents: e.g. interact with another, domain specific agent to solve a certain task.

An AI agent itself can be decomposed into several building blocks and related components:

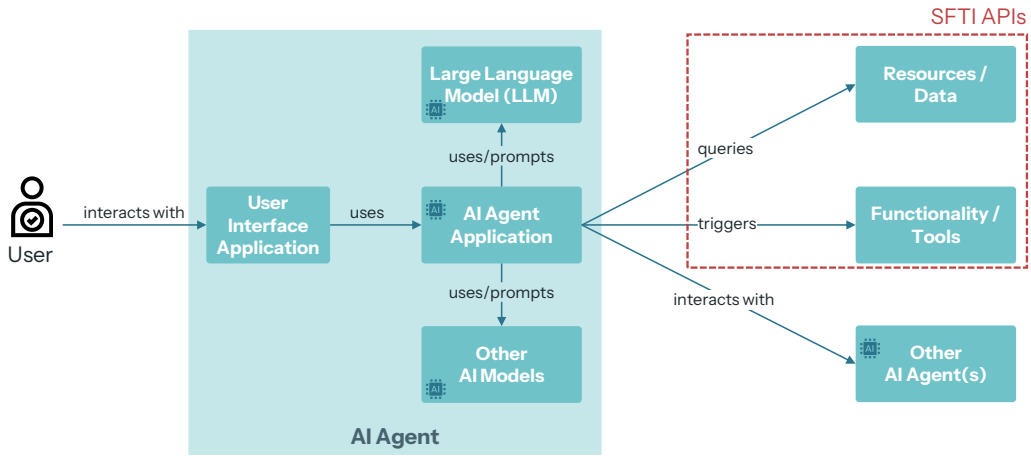

Figure 5: High-level building blocks of an AI agent and external interactions

User interface application: Provides access to functionality to the user. This could e.g. be a simple chat UI or other, more sophisticated ways of interaction. In practice most often the user interface application is part of the user facing AI agent application.

AI agent application: The main application orchestrating the interactions. It provides APIs and might be bundled with the user interface application for users or other systems to work with. It makes use of various external systems to provide its functionality.

Large language model (LLM): Typically a large language model is used by the AI agent application to analyse inputs (multi modal: text, voice, images, video) and extract data, potentially reasoning and generating output.

Other AI models: Other, more specially trained AI or machine learning models can also be used by the AI agent application for specific tasks. Possible use cases for these secondary AI models are validating the efficiency and efficacy of the primary LLM to ensure that its performance does not decay or to act as guardian (against harmful content, policy violations, data leakage, insecure or dangerous actions, hallucinations).

Resources / data: To fulfil its tasks the AI agent most likely needs context specific data / resources like e.g. documents or user related data like bank account information.

Functionality / tools: AI agents might have to trigger context specific functionality like e.g. submitting a payment or doing a specialized simulation.

Other AI agents: An AI agent might also interact with other AI agents to fulfil its tasks. Each other AI agent might again be decomposed into the building blocks listed above.

When looking at the building blocks it becomes clear that for classical (non-AI) applications resources/data and functionality/tools are accessed through (traditional) API like of course the SFTI common APIs. The question arises however if it is sufficient, to just provide AI agents access to such APIs that are currently provided. Are AI agent applications just another third-party application accessing finance APIs or are there specific aspects to be considered?

The recent developments around LLM-enhanced AI applications point into the direction that adding another layer in between can simplify integration and enhance the interaction. This additional layer provides additional context information and standardization while maintaining the key concepts already established for many open APIs (e.g. consent handling, security,…). Also, the requirement for better capability discovery arises. How can an AI agent, not knowing the exact structure and format of the classical API, be instructed on how to use them? Most recent development also introduce a “back-channel” that allows the providing side to use the capabilities of the AI agent in an interactive way (e.g. ask back

questions). In summary the situation depicted below arises: while many components of a classical (open) API architecture can be re-used, AI agent access can be facilitated with additional building blocks.

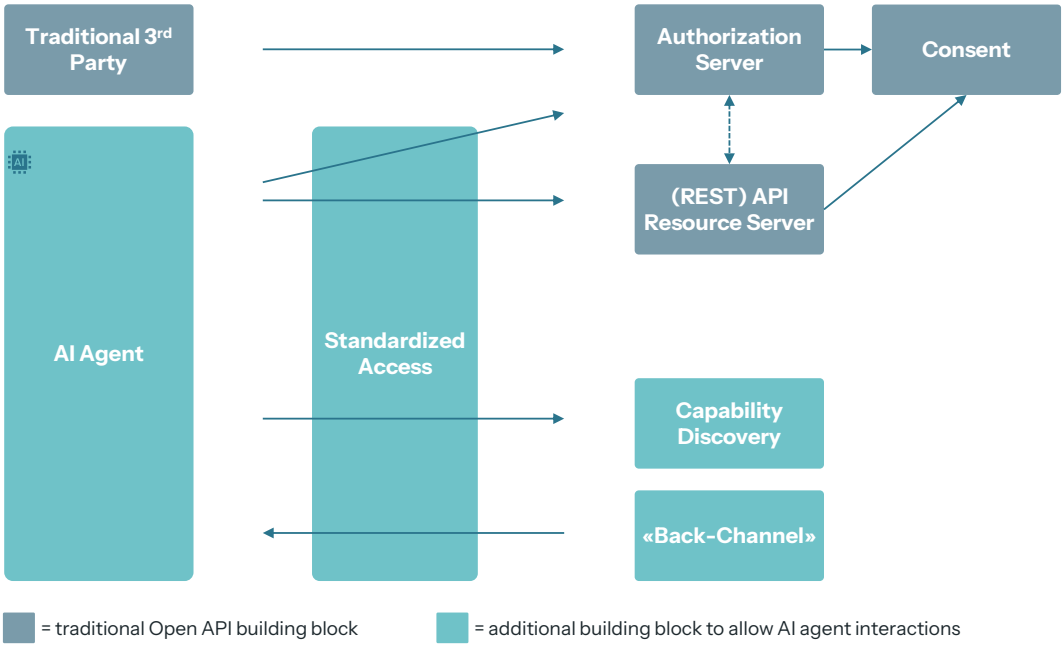

Figure 6: Additional high-level building blocks when adding AI agent interaction capabilities

This ”Back-Channels” introduce a completely new way of interaction between the provider of functionality and the consumer of such functionality. While in classical applications the consumer queries an API or triggers some functions this new way of interaction allows an actual “conversation” between consumer and provider. A provider has means to use the capabilities of the consumer side while serving requests. It could e.g. ask the consumer to provide additional information if needed (typically called “elicitation”). Or it could ask to invoke the consumer side LLM or business logic to decide on options provided or instruct it to query further external systems (typically called “sampling”).

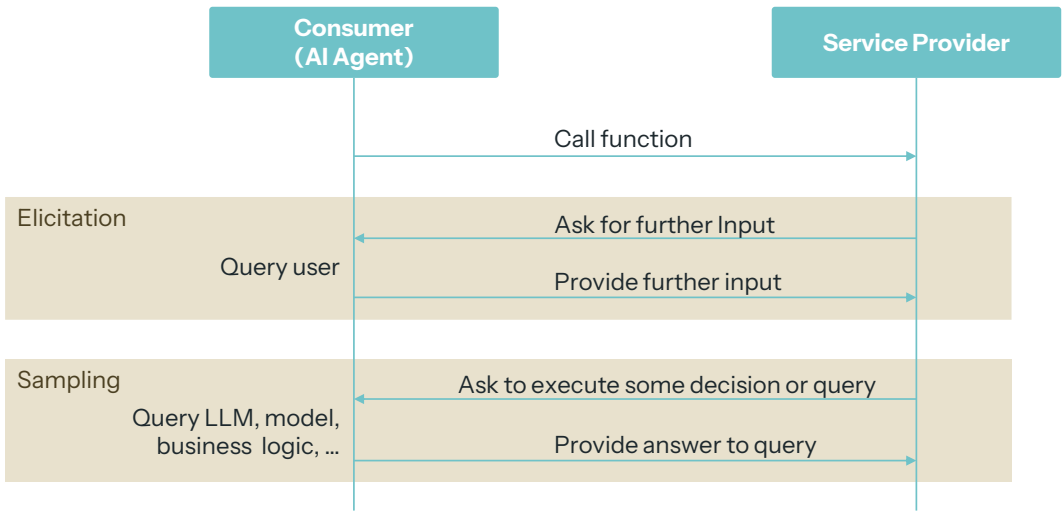

Figure 7: “Back-Channel” integration patterns

2.2 Illustrated Using the Example of a Pension Advice AI Agent

An ideal showcase for AI access to finance APIs is the cross-pillar digital pension advice. The customer need for this advice is evident and everyone can relate to the use case. Furthermore, data from several APIs is required, illustrating how in the future this should be made accessible to AI agents.

The setup depicted below is forward looking as it requires some APIs that are not yet commonly deployed. Many financial institutions are already live with the account information services (AIS) APIs (not solely but also thanks to the recent multibanking initiative). For the second pillar the relevant stakeholders have a common understanding to use the SFTI Common API Pension as API standard. The OASI (old-age and survivors' insurance) API is fully hypothetical at this moment, currently the first pillar is busy digitizing and centralizing the internal data access with project Mosar. Let's assume that all three APIs do exist.

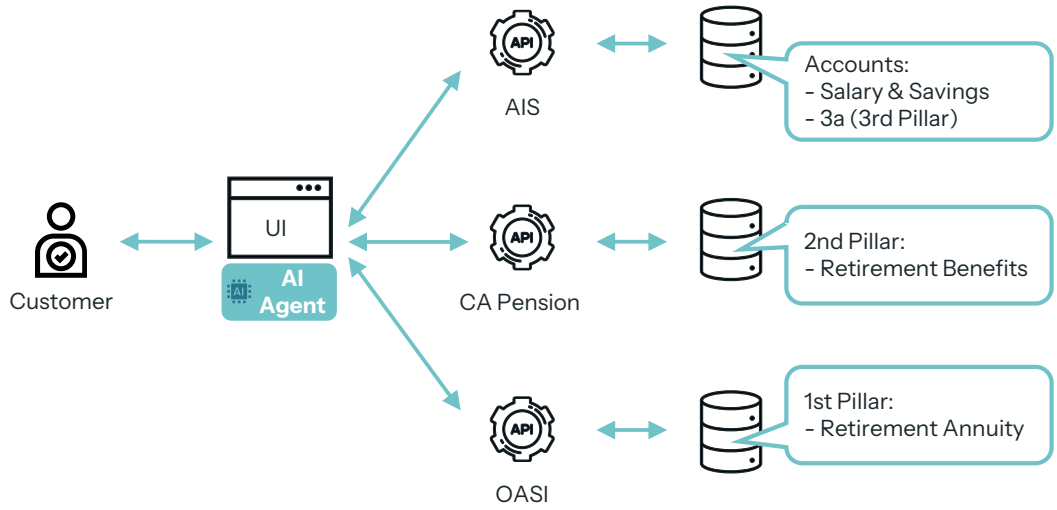

Figure 8: APIs involved in the pension advice example

The following questions illustrate what customers typically want to be answered:

- Can I still make a payment to the 3rd pillar this year? To what amount?
- Can I make a voluntary payment to the 2nd pillar? To what amount?
- Do I have gaps in the 1st pillar? How can I close them?
- How much can I withdraw for home ownership from 2nd and 3rd pillar?
- How does my overall retirement situation look like when I retire with 65?
- What do you recommend to improve my retirement situation?

Some use cases do require just one API to gather the data needed to answer the question, others need multiple APIs. But all use cases have in common, that they need the AI agent to be able to discover, understand the capability and use those APIs. Most importantly the AI agent must be able to access the APIs on behalf of the customer, the source system must know that the access is with the consent of the customer and allow it with the access scope of this customer.

3. Technical Foundations, Reference Architecture, and Guidelines

3.1 Context and Overall Architecture

3.1.1 Use Cases and Access Patterns

This white paper is written with the following use cases of the SFTI governed API standards in mind:

- **Chat**: A human is interacting with an LLM via a chat interface application. The purpose is gathering information. The LLM can ask the application to perform API calls to retrieve pertinent information.
- **Agent**: An agent is interacting with an API. The purpose is to solve a given task. The approach is to trigger the necessary actions via API to reach the goal of the task.
- **Coding**: An LLM gathers information about an API. The purpose is to build an integration towards it from another system. The use case is to simplify the process of retrieving the necessary information, then programming the integration and finally writing the documentation.

Use cases can be with a **human in the loop**, with **delegated access** or fully automatic in a **machine-to-machine** setup.

Theoretically one could also install an agent on the API provider side. This agent could be specifically trained for the use cases of the API domain. It could then be this “internal” agent which interacts with the API locally whereas the interaction to the outside becomes an agent to agent interaction. In this white paper we focus on the aspects of extending an API for AI agents. We do not cover the questions related to operating a local agent as indirect interface to the API.

3.1.2 Components

When implementing such systems that allow AI agent access the following components are needed in principle:

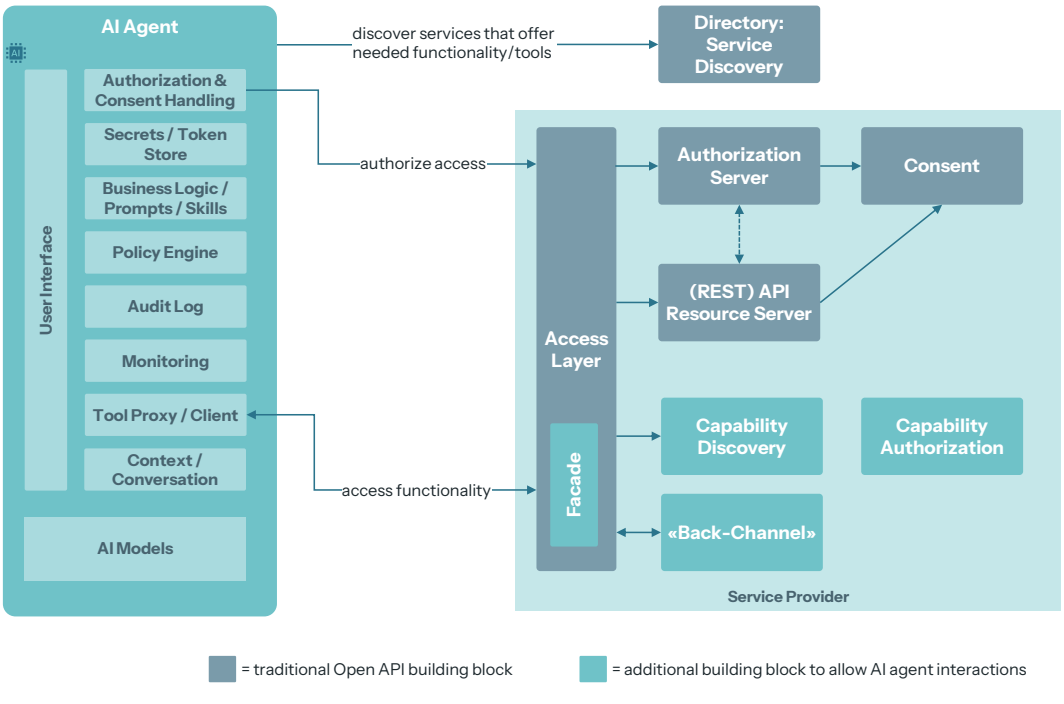

Figure 9: Main components of an AI agent and additional components for service providers

	Component	Capability Domain	Functionality
AI Agent	User Interface	Channels	Interface to interact with the AI agent (e.g. chat UI)
	Authorization & Consent Handling	Security	Handles needed authentication/authorization when accessing functionality/data at a service provider
	Secrets / Token Store	Security	Used to securely store secrets and (refresh/access) tokens needed to access functionality/data at a service provider
	Business Logic / Prompts / Skills	Business Logic & Orchestration	Agent-internal business & orchestration logic for immediate and long-running tasks. Might include predefined prompts to LLMs and other AI models and a skills directory (see https://agentskills.io/).
	Policy Engine	Business Logic & Orchestration	Engine to enforce local policies (e.g. limit usage,…)
	Audit Log	Audit	Local audit logging within AI agent
	Monitoring	Operations	Allows to monitor the agent
	Tool Proxy / Client	Integration	Local client to access remote functionality/tools
	Context / Conversation	Memory	Holds the context and conversation history
	AI Models	Conversation, Reasoning & Synthesis	Actual AI models used by the agent. Depending on the use case, this typically are large language models (LLMs) but might as well be smaller or more specialized language models (SLMs), optionally complemented by other task-specific AI or machine learning models.
Service Provider	Access Layer & Facade	API	Access layer to exposed APIs. Includes typical API gateway functionality like security enforcement, rate limiting, content-inspection,… For AI agent use cases this access layer might also perform protocol transformation (e.g. expose MCP facade for REST APIs)
	Authorization Server	Identity & Security	Authorization server that performs authentication & authorization of users/clients. Typically done using e.g. an OAuth 2.0 protocol and some defined scopes.
	Consent	Identity & Security	Consent store to register user given consent to access certain functionality and data. Might be as simple as storing given scopes but might be more sophisticated.
	(REST) API Resource Server	Various business functions	Provides the actual functionality and data exposed through APIs.
	Capability Discovery	API	Allows an AI agent to discover the provided fine-grained capabilities of a service. Includes context information on how to use a service providers capabilities (e.g. explain that a certain API can be used for a certain functionality to achieve a certain result).
	Capability Authorization	Identity & Security	Defines, what capabilities should be provided to what AI agent. Might be solved via simple authorization of scopes. But might also need more sophisticated authorization especially then using “Back-Channel”
	Back-Channel	API	Many AI agent integration protocols allow not only the usage of functionality/data but also define a way on how a service provider can “ask back” questions to an AI agent. Often also allows a service provider to hand back certain tasks to the AI agent.
External	Directory: Service Discovery		Discovery of AI agent capable services with tools/resources for a given topic. E.g. where to find information about a person’s 1st pillar pension,…

Table 1: Main components of an AI agent and additional components for service providers

3.1.3 Standards

Why is standardization of how we expose our APIs to AI agents needed? Can the AI agent not simply discover and use any exposed API?

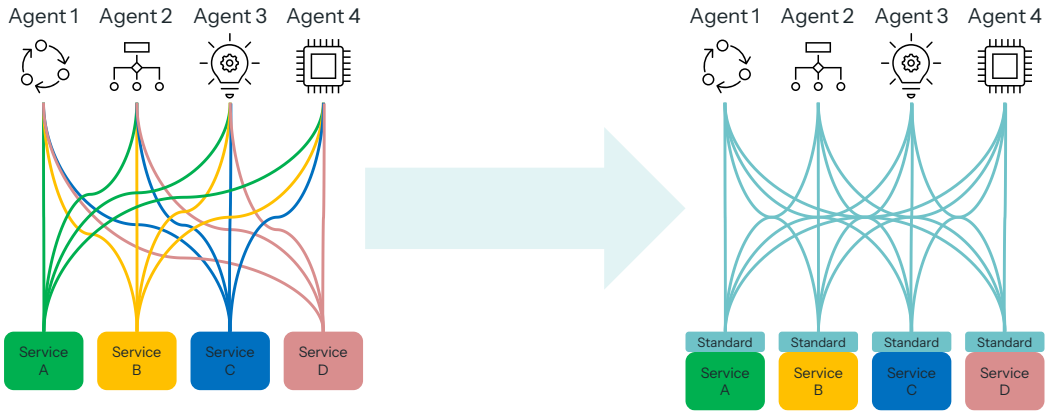

Figure 10: AI agents and service side standardisation

Standardization of the API exposures improves the following aspects for the AI agent:

- **Service discovery**: When a service always looks the same, then it is easier for an agent to identify it as such.
- **Reduced ambiguity**: Providing context to the agent in a standardized way helps it to know what offering a service covers.
- **Initial setup/connection**: A standard provides efficiency and clarity on how the connection from the agent to the service can be established and how the interaction is organized.
- **Authentication**: For services which offer user specific resources it is important to know how that user can be authenticated by the service.
- **Targeted service usage**: Having a clear description of each resource a service provides helps the agent to make optimal use of the service without the need of trying it out.
- **Reduce load on AI model**: The standardization shall guide the agent to make economic use of the service so it can operate with less tokens and less calls to achieve its goal.

In the context of AI agents and APIs there are three distinct types of connections, for all of which different standards have emerged.

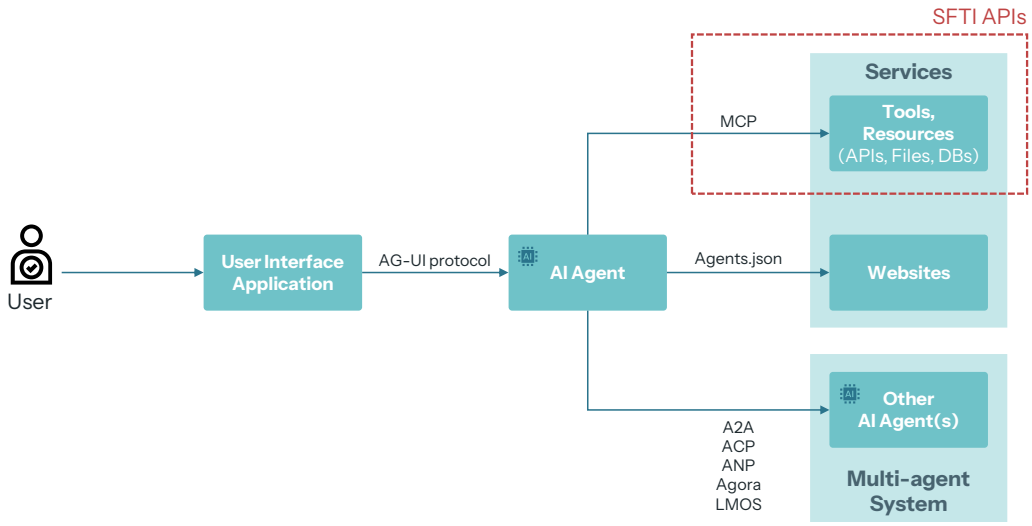

Figure 11: Connection types of the AI agent

Agent – Service: For accessing services two standards have been defined

- **Model Context Protocol (MCP)** – an open-source standard for connecting AI applications to external systems. It helps AI applications to connect to data sources (e.g. local files, databases), tools (e.g. search engines, calculators) and workflows (e.g. specialized prompts) – enabling them to access key information and perform tasks. https://modelcontextprotocol.io/docs/getting-started/intro
- **Agents.json** – is a JSON-based schema designed for websites to explicitly declare how autonomous AI agents can interact with them. It contains metadata about the site, what kinds of actions or "intents" are allowed (for example, reading content, submitting data, or performing searches), and any authentication or rate-limit requirements. https://github.com/wild-card-ai/agents-json?tab=readme-ov-file

Note: We do not cover proprietary / direct integrations in this white paper. But we have included an example in the appendix 5.2 for your comparison.

User Interface Application – Agent: In the domain of connecting a user interface application to an agent there is just one open protocol

- **Agent-User Interaction (AG-UI) protocol** – standardizes how agent state, UI intents, and user interactions flow between model/agent runtime and user-facing frontend applications. https://docs.ag-ui.com/introduction

Agent – Agent: And finally for the agent to agent connections there are multiple protocols through which they can discover each other, collaborate, delegate and manage shared tasks

- **Agent2Agent (A2A) protocol** – open standard designed to enable seamless communication and collaboration between AI agents. Originally developed by Google and now donated to the Linux Foundation. https://a2a-protocol.org/latest/
- **Agent Communication Protocol (ACP)** – open protocol for agent interoperability that solves the growing challenge of connecting AI agents. Originally developed by IBM Research and now merged into the A2A protocol. https://agentcommunicationprotocol.dev/introduction/welcome

- **Agent Network Protocol (ANP)** – goal is to be “the HTTP of the agentic web era.”. As such, it employs HTTP for data transport, W3C DID (Decentralized Identifiers) and JSON-LD (JSON for Linked Data) for data formatting. Three layer architecture similar to LMOS. https://www.agent-network-protocol.com/
- **Agora** – Agora provides a common way for different agents – potentially built on different frameworks or models – to communicate using both natural language and structured data. It supports multi-round conversations, protocol versioning, and decentralized interactions, but it leaves authentication, security, and coordination mechanisms to higher layers. https://agoraprotocol.org/
- **LMOS protocol** – aims to usher in an Internet of Agents (IoA), a multi-agent ecosystem on an internet scale. Developed by the Eclipse Foundation. Similar to ANP, its structured architecture consists of three layers (Identity and security, transport protocol and application protocol layer). https://eclipse.dev/lmos/docs/introduction/

For our analysis on how AI agents can be supported in accessing SFTI APIs we can **limit our focus on the MCP**. Although Agents.json is also an agent to resource standard – it is not relevant to us, as it meant for websites – not APIs. When it comes to authentication, we will also need to analyse multi agent setups.

3.1.4 Model Context Protocol (MCP)

Anthropic published the Model Context Protocol in November 2024 as an open standard. It has quickly gained wide adoption and established itself as the unrivalled standard. In this chapter, we provide a brief introduction to the specification based on the version valid in autumn 2025. However, the standard is evolving quickly.

Before introducing the MCP in detail, it is important to distinguish between the information provided by a traditional API specification and the additional context required by AI agents. While good API specifications should provide a good overview of the business context they often primarily describe the structural contract of an interface: endpoints, request and response schemas, parameters, and security mechanisms. They are designed for deterministic integration by developers and applications. MCP does not replace these specifications. Instead, it augments them with a standard way to provide agent-oriented context that focuses on capabilities, intent, and interaction patterns, enabling AI agents to understand what can be achieved with a service, how available functions relate to tasks, and how interactions can be orchestrated dynamically. In this sense, MCP builds on existing APIs by adding a semantic and interaction layer tailored to the needs of agentic systems.

MCP defines the following entities:

- **MCP host**: The AI application that coordinates and manages one or multiple MCP clients
- **MCP client**: A component that maintains a connection to an MCP server and obtains context from an MCP server for the MCP host to use
- **MCP server**: A server that provides context and interface of a resource to MCP clients

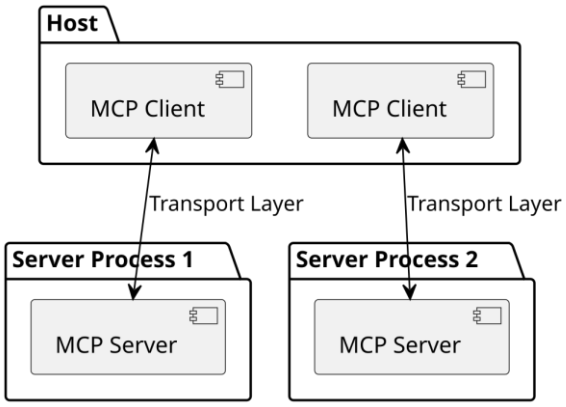

Figure 12: Entities defined by MCP. Source: https://en.wikipedia.org/wiki/Model_Context_Protocol

It is also worth mentioning that MCP servers can expose local resources. For example, they can expose the file system or resources of any application, such as the calendar and inbox of Outlook. As this white paper focuses on exposing SFTI APIs, it does not cover such local MCP servers.

The terminology in this chapter is the official one used by the MCP definitions. It is different from our agnostic definition in 3.1.2 Components. The following straightforward mapping applies:

- MCP host: AI agent
- MCP client: ToolProxy / Client (component inside the AI agent)
- MCP server: Facade, Capability Discovery & Back-Channel

Layers

On the technical level MCP consists of two layers:

- **Data layer**: Defines the JSON-RPC 2.0 (JavaScript object notation - remote procedure call) based protocol for client-server communication, including
  - **Lifecycle management:** Connection negotiation and management
  - **Server features**: The core primitives a server offers (tools, resources, prompts)
  - **Client features**: Enables servers to ask the client (samples, user input, logs)
  - **Utility features**: Additional capabilities (notifications, progress tracking)
- **Transport layer**: Defines the communication mechanisms and channels that enable data exchange between clients and servers, including transport-specific connection establishment, message framing, and authorization. MCP supports two transport mechanisms:
  - **Stdio transport**: Uses standard input/output streams for direct process communication between local processes on the same machine
  - **Streamable HTTP transport**: Uses HTTP POST for client-to-server messages with optional Server-Sent Events for streaming capabilities. It enables remote server communication and supports standard HTTP authentication methods including bearer tokens, API keys, and custom headers.

Conceptually the data layer is the inner layer, while the transport layer is the outer layer abstracting the communication details from different transport mechanisms away from the data layer.

Data Layer – Server Features

MCP primitives are the most important concept within MCP. They define the context information and the actions which clients and servers can offer each other.

MCP servers can expose three core primitives:

- **Tools**: Executable functions that AI applications can invoke to perform actions (with user approval, including write operations; e.g., file operations, API calls, database queries)
- **Resources**: Data sources that provide input information to AI applications (e.g., file contents, database records, API responses)
- **Prompts**: Preformulated templates that help structure interactions with language models for specific tasks (e.g., system prompts, few-shot examples)

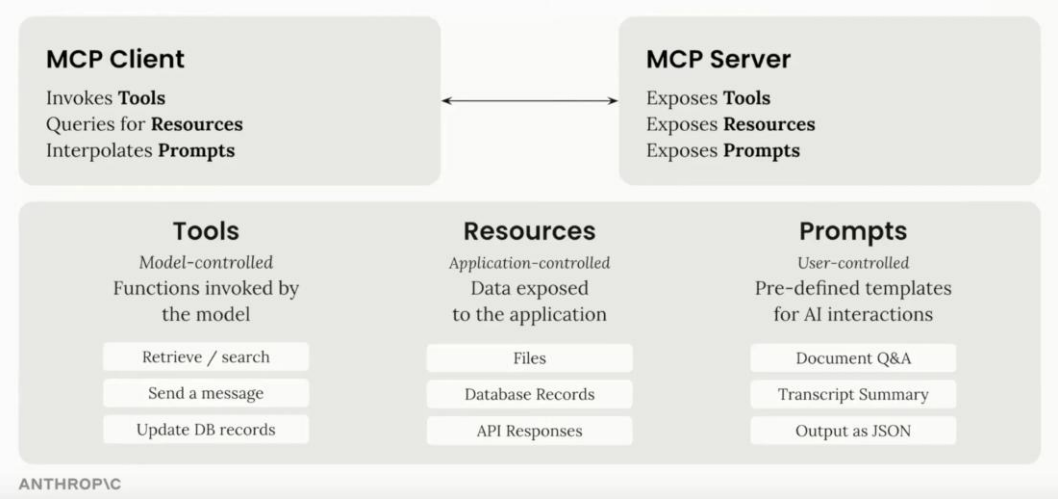

Figure 13: MCP server core primitives. Source: https://www.anthropic.com/

Each primitive type has associated methods for discovery (*/list), retrieval (*/get), and in some cases, execution (tools/call). MCP clients will use the */list methods to discover available primitives.

Data Layer – Client Features

MCP also defines primitives that clients can expose (named “Back-Channel” above). These primitives allow MCP server authors to build richer interactions.

- **Sampling**: Allows servers to request language model completions from the client’s AI application. This is useful when servers’ authors want access to a language model, but want to stay model independent and not include a language model SDK in their MCP server.
- **Elicitation**: Allows servers to request additional information from users. This is useful when servers’ authors want to get more information from the user, or ask for confirmation of an action.
- **Logging**: Enables servers to send log messages to clients for debugging and monitoring purposes.

3.1.5 Multi-Agent Systems

In this paper, we focus on the exposure of services to AI agents. However, real-world use cases can involve multiple agents to solve a task. What needs to be considered from the service providers point of view when there is a cascade of agents involved instead of just a single agent?

Communication protocol: Agent to agent communication is not using MCP. Instead, other protocols are used such as A2A or ACP. The agent calling the service will still use MCP, and therefor this has no impact on the service provider.

True principal: In a one agent setup, the MCP server is called by the user-facing agent. In a multi-agent setup, the MCP server is called by an intermediate agent. The MCP server must make authorization decisions based on the effective user and not on the immediate caller. This is why the MCP server in any case needs a user authentication. There are two ways how the agent calling the service can obtain the needed bearer token: either it solicits the user for an authentication flow (performed out of band) or the bearer token when it already exists for that service is propagated downstream. Either way for the MCP server a valid bearer token will be presented and the authorisation can be performed accordingly. These approaches are a relevant concern as downstream agents, potentially not from the same trust domain, are coming in possession of bearer tokens. The user must be in control of which agent is given which access. An alternative approach is to use an orchestrator agent which offers a helper function that forwards the calls to MCP servers, this is then the only agent to see the bearer token. Independent from the multi-agent setup this is outside of the direct influence of the MCP server. What can be done on MCP server side is to block certain agents in the entry layer.

Idempotency: With more agents in the loop there are higher chances of retries, mis-ordered calls and speculative execution. When possible, an agent should detect repeated requests and return the same result – or reject them as duplicates for operations that have write character.

Data protection: Data minimization becomes even more important as the data flows through multiple agents. Currently the inter agent protocols (and also MCP) do not offer encryption mechanisms to encrypt sensitive message parts so that only the user facing agent can read it.

Back-Channel: Inter agent protocols such as A2A or ACP currently don't support elicitation. This means a request for user input formulated in the MCP cannot be forwarded directly by an intermediary agent to the user facing agent. Potentially the intermediary agent will reformulate the question and forward it via the conversational channel in the direction to the user risking altering the intent or simply ignore the elicitation. It will then convert, formulate or hallucinate the elicitation response to the MCP server. In short elicitation today results in degraded behaviour in multi agent setups. Hopefully this is just a temporary shortcoming and inter agent protocols will close this gap soon. When it comes to sampling, this doesn't change in multi-agent setups, its purpose is just for the MCP server to make use of the AI model of the agent, this is possible by using the model of the intermediary agent. Forward sampling to the user facing agent is unnecessary.

Further topics arise in multi-agent scenarios, but the following ones we don't elaborate as they are not relevant from service provider perspective:

- Multi hop latency issues and resulting degraded user experience
- Observability and tracing across the agent chain

3.2 Security & Data Protection

3.2.1 Threats & Mitigations

One can argue that the API-related threats have already existed before the APIs were extended by the addition of MCP servers. So, what changes with MCP?

- We will have a new kind of API clients, they are expected to not go through a classic integration cycle, instead they might integrate ad hoc.
- Functions will be called for more surprising and creative use cases than today. For example because of misunderstood capability of the API.
- Input data will be composed from less reliable sources (user prompts, web content and hallucinated input).
- Anomaly detection becomes more challenging because differentiation between human and automated (now AI) behaviour is becoming more difficult.

On top of this new usage patterns the deliberate, malicious attacks remain. Potentially they can be performed more efficiently via agents. What are their goals?

- Exfiltrate sensitive data
- Abuse capabilities (state changes, transactions)
- Disrupt availability or run up costs

Which are the MCP specific attack surfaces and threats we need to consider?

Note: We are not discussing threats which are inherent to exposing APIs. Those remain and probably the related remedies should be hardened because of the new usage patterns mentioned above. Also, we focus on threats which can be mitigated on the server side, we do not analyse client side threats.

Authentication and Authorization – Privilege Escalation

- When an MCP server performs an action triggered by a user's request, there is a risk of a "confused deputy" problem. If the server is not implemented correctly, a user could gain access to resources that should not be available to them but that are available to the MCP server.
- Mitigation: The API must be called on behalf of the user by propagating the identity to the API. MCP defines authorization using OAuth. Do not call the APIs with a technical user, that would burden the MCP server with fine grained access control for which it is missing the business logic.

Supply Chain Risks & Tool Injection

- MCP servers are composed of executable code and malicious code could be injected. A cloud service based MCP server could act as man in the middle or extract/inject data.
- Mitigation: MCP servers must be developed using the same pipeline and security measures as with any other development. Including dependency management and scrutiny of cloud service providers if applicable.

Prompt Injection

- MCP servers pose inherent security risks due to their ability to execute code themselves and also as they execute commands via API calls. An unintended action could be triggered by the LLM's interpretation of a prompt or by a complex, obfuscated prompt copy/pasted from a dubious source. E.g. the obfuscated prompt could create two users in the target system instead of one.
- Mitigation: In case that received input is used in local execution, that input needs to be validated and sanitized. Actions to be called by the MCP servers should always be confirmed by the users (e.g. by Elicitation) or restricted to reduce risk to an acceptable level.

Vulnerability Management

- Since MCP servers are code, they may have vulnerabilities just like any other software.
- Mitigation: It's critical to include them into the standard vulnerability management process.

Sampling

- Malicious MCP servers may also try to exploit the MCP sampling functionality, where MCP servers can request MCP clients to use an LLM to complete a request and fish for information via this MCP feature.
- Mitigation: Since in this paper we discuss MCP servers (which we control), we assume that “our” MCP servers will not abuse this feature. Mitigation is actually on the MCP client side. In case you do implement this feature server side, be aware that the client might show the completion request to the user, might allow the user to reject/modify it and implement rate limits and timeouts which result in completion failure.

Information Leakage

- Information can be leaked unintentionally by verbose error messages or stack traces emitted back to the model. It can also be leaked indirectly by inference based on error/success responses on queries. Another leak is not in the direction of the MCP client, but it is still relevant: Information can be leaked by writing secrets or personal data into the MCP log.
- Mitigation: Limit content of error/response messages to the necessary. Do not persist full request/response in the log.

Excessive Permission / Excessive Information Disclosure

- When a single, powerful endpoint is exposed to retrieve the data or the endpoint is called with an overly powerful user, then the MCP client will receive much more information than what is needed for the actual task/question at hand. This can lead to unwanted information disclosure.
- Mitigation: Create multiple, clearly scoped functions and resources to limit unnecessary information expose.

Authentication Token Exposure

- To perform an authenticated request the bearer token is received by the MCP server and forwarded to the API. These OAuth tokens are an attractive target for credential theft.
- Mitigation: See chapter 3.2.2.

Phishing

- Users (or agent-based systems) can be tricked into connecting to an imposter MCP server. In the process they expose their credentials to the phishing MCP server.
- Mitigation: Since we don’t control the client side we cannot enforce any measures. But we must make our MCP server easily discoverable so that users don’t start looking in the wrong places finding imposters. Ideally our server can prove to be the correct server with signed certificates or potentially with emerging self-sovereign identity (SSI) based decentralized identifier (DID) solutions.

Excessive Load

- Malicious clients could attempt to overload the system in order to make it unavailable. They could also try to drive up costs for the service owner, for example when AI capabilities are exposed the cost per answer can be relatively high.
- Mitigation: The usual DDoS defence strategies should be applied for the first type. For the latter quotes need to be installed and enforced.

Inefficiency Risk

- A inherent threat is that the agent's AI model needs an excessive amount of tokens to process a lengthy response running up costs. Or that many tool calls are needed to complete a task resulting in latency for the user and again increased token usage. Or that the conversation becomes too large to remain in the agent's context window and thus loosing context partially. Also hallucinations increase the more tokens we use.
- Mitigation: Identify clear use cases, model them as resources (think of "jobs to be done") and tailer the endpoints accordingly which must output short and concise responses. When needed guide agents to more specific requests through elicitation before giving broad responses.

3.2.2 Access and Consent Handling

To access private or paywalled services the AI agent needs an identity with the according rights. Typically for the access to be granted, the user will need to perform a login and give consent to generate a token with which the agent can use the service. Therefore, the access to the service will be on behalf of that user and this will define the scope for what the service can deliver.

As also true for APIs the service provider can decide whether the service can be consumed by any AI agent or whether it is a closed ecosystem with permission-based access (e.g. by accepting requests only from certain IP addresses). Potential agent restrictions are not part of the MCP, they need to be implemented in the existing security dispositive of the service provider.

Once a MCP connection between MCP client and MCP server is established the MCP server can make use of the "Back-Channel" for example via elicitation This requires that the MCP client has declared support for it during initialisation. The MCP server can then use resources and draw information from the MCP client as long as the connection is valid.

Technical Considerations

On transport layer for remote servers, MCP relies on HTTP transport. When accessing restricted endpoints various authentication standards could be used. It is important to avoid weak authentication standards. The recommended authentication is OAuth 2.1 (Open Authorization).

Following the authorization flow the user will be presented with a login screen in a browser to grant the required permissions resulting in a bearer token sent to the MCP client. This bearer token will then be used by the MCP client in the HTTP header to authenticate requests to the MCP server.

A compromise of an MCP host in an AI agent containing multiple MCP clients can give access to multiple services via the persisted Bearer tokens. In a multi-agent setup downstream agents might also hold Bearer tokens and accept requests from the compromised upstream agent resulting in access to further services.

For the SFTI APIs there is a dedicated common API work stream covering the security, access and consent topics. Details can be found here: https://github.com/swissfintechinnovations/ca-security/wiki/Consent-Management.

Security best practices for implementing the MCP server:

- **Do not implement token validation or authorization logic by yourself**. Use off-the-shelf, well-tested, and secure libraries for things like token validation or authorization decisions. Doing everything from scratch means that you're more likely to implement things incorrectly unless you are a security expert.
- **Use short-lived access tokens**. Depending on the authorization server used, this setting might be customizable. We recommend to not use long-lived tokens - if a

malicious actor steals them, they will be able to maintain their access for longer periods.

- **Always validate tokens**. Just because your server received a token does not mean that the token is valid or that it's meant for your server. Always verify that what your MCP server is getting from the client matches the required constraints.
- **Store tokens in secure, encrypted storage**. In certain scenarios, you might need to cache tokens server-side. If that is the case, ensure that the storage has the right access controls and cannot be easily exfiltrated by malicious parties with access to your server. You should also implement robust cache eviction policies to ensure that your MCP server is not re-using expired or otherwise invalid tokens.
- **Don't expose tokens**. The tokens mustn't be exposed in the body of any request or response. In general, it also mustn't be exposed to any AI model.
- **Enforce HTTPS in production**. Do not accept tokens or redirect callbacks over plain HTTP except for localhost during development.
- **Least-privilege scopes**. Don't use catch-all scopes. Split access per tool or capability where possible and verify required scopes per route/tool on the resource server. When possible, use Rich Authorization Requests to minimize the consent a user needs to give (see https://datatracker.ietf.org/doc/rfc9396/).
- **Don't log credentials**. Never log authorization headers, tokens, codes, or secrets. Scrub query strings and headers. Redact sensitive fields in structured logs.
- **Separate app vs. resource server credentials**. Don't reuse your MCP server's client secret for end-user flows. Store all secrets in a proper secret manager, not in source control.
- **Return proper challenges**. For HTTP 401 responses, include the WWW-Authenticate header with Bearer token, realm, and resource_metadata so clients can discover how to authenticate.
- **DCR (Dynamic Client Registration) controls**. If enabled, be aware of constraints specific to your organization, such as trusted hosts, required vetting, and audited registrations. Unauthenticated DCR means that anyone can register any client with your authorization server.
- **Multi-tenant/realm mix-ups**. Pin to a single issuer/tenant unless explicitly multi-tenant. Reject tokens from other realms even if signed by the same authorization server.
- **Audience/resource indicator misuse**. Don't configure or accept generic audiences (like api) or unrelated resources. Require the audience/resource to match your configured server.
- **Error detail leakage**. Return generic messages to clients, but log detailed reasons with correlation IDs internally to aid troubleshooting without exposing internals.
- **Session identifier hardening**. Treat Mcp-Session-Id as untrusted input; never tie authorization to it. Regenerate on auth changes and validate lifecycle server-side.

3.2.3 Data Protection

Regarding data protection we structure the analysis along the involved parties:

- Data protection measures of the **MCP server**
- Potential data leakage by the **MCP host**
- Data privacy applied by the **LLM provider**
- **Mutli-agent** scenarios

MCP Server

First of all, the MCP server can support data protection by only providing the data which is required for a certain task. This means exposing dedicated, data-frugal endpoints. This can translate in adding dedicated endpoints to the underlaying API in case that the API currently has verbose endpoints. It should be noted, that when exposing many data-frugal endpoints, that the MCP host still might have access to verbose information through the sum of endpoints and it is in the control of the agent which information is collected and used.

Server side caching of data and/or session states might be implemented for better performance and improved responses. This opens the possibility of data leakage across conversation, long-term persistence of sensitive data and also makes the MCP server a target for scraping attacks.

Logging the traffic going through the MCP server can persist sensitive data. Ideally only non-sensitive data should be logged. And a centralized log server should be used in order to avoid long term persistence of data on the MCP server.

MCP Host

The direct data privacy issues are related to the host itself. Is it running locally or remotely? Who has access to the data it processes? What data is persisted and how is it protected?

Usage scenarios can leak data directly to third parties. When a MCP host has multiple MCP clients connected to multiple MCP servers, then it might get sensitive data from one MCP server and further use it in a request to another MCP server (typically another third party than the origin of the sensitive data). The user might not intend or expect this data sharing.

LLM Provider

In general, LLM provider love data and are keen to store and use received data for further improving their LLM. This data can be the conversation itself but also any data received during the conversation.

Typically, in cases where the LLM provider also provides the UI, especially in a paid context, one can instruct the provider, that the data is private to the user and it may not be used for other purposes than the actual conversation. Currently the terms and conditions of the providers might state, that the privacy setting influences how they treat data received via the user interface. This leaves ambiguity if this setting also applies to the data received via other channels than UI, for example backend integrations.

Presumably this is just a transitory issue and the terms and conditions simply lag behind the technical development. Therefore, it is just the user, which needs to disable the “Improve the model for everyone” (naming ChatGPT) setting.

Multi-Agent Scenarios

In the multi-agent scenario, the data protection concerns remain overall the same as already mentioned above. Just that the path the data can take and the parties being involved in processing the data becomes even more complex.

Note: We do not offer actionable recommendation for the issues identified for the MCP host and the LLM provider as they are outside of the control of the realm of the entity offering the MCP server. But they are relevant in the sense that one needs to decide which data endpoints one wants to offer via an MCP server.

3.3 Operations

3.3.1 Exposing the MCP

Organizations adopting MCP should treat exposed MCP servers with the same rigor as critical API infrastructure – ensuring secure exposure, managed findability and proper client onboarding.

Exposure

The MCP server is deployed behind the existing web access layer, such as a web application firewall (WAF) and API gateway. It shall reuse the existing security controls including TLS termination, traffic inspection, and authentication enforcement. Exposure is achieved by making the MCP server reachable over a chosen transport. For remote servers this is commonly a HTTP-based bidirectional channel. Operationally this involves:

- Ensuring the MCP server process (or adapter) is deployed in a reachable execution environment.
- Providing the necessary connection details to MCP clients (host, protocol, port, authentication method).

Modern API gateways already support exposing MCP servers in a managed manner. Analogue to API exposure, such gateways allow to set and enforce rate limits, client quotas and IP filtering. They can also validate request authorization.

Findability

The server is not inherently findable by MCP clients. In general, the MCP specification does not currently define how MCP servers can be found (e.g. directories or service registries). Therefore, organizations must implement their own operational convention for publishing MCP servers. The approaches differ when exposing internally or externally:

- Static exposure information can be distributed directly in controlled environments (e.g. IDE extensions, local tools).
- Internal centralized service registry (e.g., Kubernetes Service Catalog). This allows internal developers and MCP clients to resolve MCP servers dynamically.
- API gateway or developer portal listing. If existing this acts as a canonical registry where MCP servers can be registered similarly to REST APIs. Access tokens / keys can be provisioned alongside standard API credentials.
- Public registries (emerging pattern). While not yet standardized, there is movement toward public registries (e.g. https://getmcp.io/registry/) where third-party MCP servers can register their capabilities. These registries function similarly to package indexes.

Client Registration

MCP servers do not inherently require client registration based on the protocol alone. However, in operational practice, access control is essential. Client registration is optional but strongly recommended depending on the deployment context:

- Private or local MCP servers: No registration needed. This applies to embedded MCP servers shipping inside applications and local integrations (stdio). Registration is unnecessary. The client can connect freely as long as the local environment permits it (e.g., process-level permissions).
- Public or semi-public MCP servers: Requires registration. In general MCP endpoints exposed over a network. Clients typically must authenticate (API keys, OAuth2, mTLS). Authorization rules can restrict which tools or resources a given client may access. Client registration allows organizations to track usage, audit

access, and manage lifecycle. For multi-tenant or regulated environments: Strong registration controls are needed. In environments such as banking identity-bound client registration is required.

3.3.2 Operating the Solution

While design and exposure define what the MCP server offers and how it can be accessed. Operations define how reliably these capabilities run, scale, and evolve over time. Operating an MCP-based integration layer requires careful planning and continuous oversight. Agent driven load patterns may be unpredictable, making resilience, scalability, and observability essential. Downstream APIs must be protected from overload, and robust governance ensures that capabilities evolve safely and transparently. Through thoughtful operational design, organizations can ensure that MCP servers function reliably, securely, and efficiently – supporting both humans and intelligent agents in a scalable, future-proof architecture.

Load Patterns

When exposing existing APIs via MCP, consumption patterns can differ significantly from conventional client behaviour. MCP clients – especially AI agents – may generate dynamic, bursty, or exploratory workloads:

- Bursty, non-linear workloads: AI-driven clients may execute rapid sequences of tool calls or resource fetches. Unlike human-driven or business-system-driven APIs, request volume can spike suddenly during complex reasoning tasks and create cascading downstream API calls. The system must be designed to be able to handle such bursts.
- Load multiplier: An MCP server that wraps legacy or unadapted APIs acts as a load multiplier. Each tool invocation may trigger multiple API calls or transformations. The API endpoints need to be adapted to the exposed capabilities.
- Latency sensitivity: MCP clients often work interactively. Delays longer than a few hundred milliseconds can degrade user experience or lead to timeouts in agent workflows. Round-trip latency per capability and queuing/throttling delays must be minimized for all load situations.

Resilience

Resilience ensures that the MCP server continues to function – even when downstream systems do not. This offers MCP clients a stable and defined behaviour:

- Fail-gracefully: Downstream failures should be detected quickly and structured, machine-readable errors need to be returned to the clients. This helps to avoid cascading retries that amplify failures. Provide alternative responses or fallback capabilities where possible.
- Graceful degradation: When full functionality is not available one can disable certain capabilities temporarily, provide clients with reduced capability sets and even communicate the current capability availability through metadata. This allows agents to adapt dynamically.
- Downstream protection: To prevent overwhelming underlying systems with AI loops we can implement circuit breakers for unhealthy endpoints, apply exponential backoff and retry policies and degrade non-essential capabilities under stress.
- Redundant deployment topologies: For the resilience of critical MCP servers themselves we can create a high availability setup by operating multiple MCP server instances.

Performance

Operating MCP servers efficiently demands a combination of proactive and reactive performance strategies. These strategies are in line with best practices for assuring performance of service exposed via the Internet:

- Horizontal scalability: Due to bursty AI-driven traffic, horizontal scaling is typically more effective than vertical scaling. Ideally based on autoscaling features of the cloud technology.
- Rate limiting & quotas: To ensure fairness and stability apply per-client rate limits, define quota tiers based on client type and communicate the limits to MCP clients so agents can adapt their behaviour. Considering multi-agent scenarios you might need to setup per-principal and per-agent quotas.
- Avoid unbounded execution: Tool calls can result in infinite loops for example when the MCP server asks for additional information via sampling. Furthermore, in multi agent systems unexpected recursions or deadlock scenarios can occur. To avoid these scenarios, one needs to limit the call stack depth and incorporate timeouts.
- Caching strategies: For resource reads responses can be cached to reduce load on back-end APIs. In case of capability trees also metadata can be cached.
- Load testing: Avoid performance issues going to production. Perform load testing when a new version of the MCP server, typically with a new capability schema, is ready for go live.

Observability & Monitoring

Modern operational standards require complete insight into system behaviour as well as reliable mechanisms for detecting, diagnosing, and responding to issues. MCP's capability-oriented runtime introduces new requirements across both observability and monitoring, which together ensure performance, reliability, and security:

- Metrics: We suggest to track connection counts, invocation per tool/resource, downstream API call counts, token usage or payload size, error rates and failure modes per capability, latency per client and capability. In general, one should also observe distribution, not just averages.
- Logging: As much as permitted by data protection requirements one should log all MCP events including authentication, authorization, capability invocations, and responses with detailed context including user identity, timestamps, parameters, results, input and output schema references, error codes and downstream system statuses.
- Centralization: The metrics data can be integrated into the existing API monitoring stack or into an overarching OpenTelemetry or similar frameworks which can unify and centralize tracing. The MCP server should have the capability of sending the logs to standard centralized logging servers. The advantage of centralized logging and telemetry systems is that it enables end-to-end operational monitoring, tracing and analysis. When no such centralized systems exist, then it should be able to monitor and persist the traces locally. Having these traces enables thorough investigation into system actions if a problem occurs or is suspected.
- Surveillance: The log data should also be sent to the security information and event management (SIEM) system for security surveillance. The SIEM can then integrate the MCP events into its real-time correlation rules for security event detection and automated incident response workflows.
- Alerting & SLOs: Define service level objectives such as 99th percentile latency, capability availability thresholds, maximum tolerated downstream error rate. Trigger alerts on deviations before end users notice them.

Lifecycle Management

Like any production service, MCP servers require stable deployment and change management processes.

- Versioning: All involved components should be versioned for traceability. This includes the MCP server itself, the code exposing the tools and resources, configurations, documentation and sample payloads.
- Configurations: Externalize all configuration with environment-specific overrides to support the use automated deployment pipelines.
- Rolling updates: For high availability services use rolling deployments to ensure no downtime during the updates.
- Capability lifecycle governance: Define processes for introducing new capabilities, deprecating legacy API wrappers, migrating tools as backend APIs evolve and archival or deprovisioning workflows.

Operational Governance and Compliance

Last but not least some considerations regarding oversight:

- Auditability: Maintain audit trails for client interactions, configuration changes and sensitive operations executed via tools.
- Policy enforcement: Use automation to enforce data retention policies, least-privilege access and geographic or latency-based routing rules.
- Compliance integration: Integrate MCP components and operations with enterprise frameworks such as ISO 27001 and SOC 2. In case that you also operate AI components, then you should also consider ISO 42001.

3.4 Limitations of MCP

While MCP solves the integration of functionality and data into AI agents (and LLMs directly) very well, there are also some limitations and drawbacks that need to be mentioned:

- When naively integrated into an AI agent or LLM the MCP tool and resource definitions may occupy a significant portion of the available context window. I.E. when an AI agent or LLM discovers all the capabilities of all its MCP tools/resources this amount of data can become so large that processing this information within the context can become slow and expensive (token usage). However, this is not an MCP flaw per se but rather an issue about how MCP is integrated into an AI agent. There are already developments to further abstract tools/resources into "skills" that provide a higher level (and shorter) description and allow an AI agent to just load more detailed MCP based capability information lazily when used within a corresponding "skill". See https://agentskills.io/ for further information about this approach.
- While MCP allows an AI agents to explore functionality dynamically in many use cases it's already clear what exact functionality an AI agent will need to execute often in a rather static way. So from an AI agent implementers perspective it may make sense to integrate traditional APIs directly in that case. From a service providers perspective it will most certainly make sense to provide both
    - a traditional API to integrate with AI agents (and third parties in general) in a deterministic way
    - an MCP based access that allows rather explorative access to data and functionality
- In many cases just providing a simple pass-through MCP layer wrapping existing APIs proved to be not sufficient for (the currently still very limited) AI agents to properly understand them. It often proved necessary to rethink what use cases might occur to AI agents and provide MCP tools that better guide them in their tasks to be achieved.

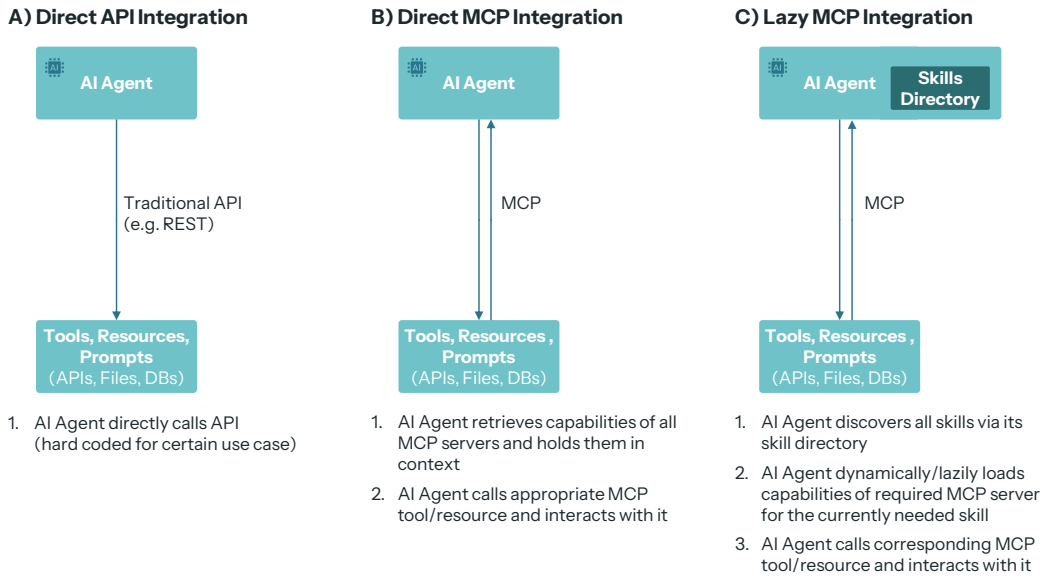

Figure 14: AI agent integration options

4. SFTI Recommendation and Proposed Next Steps

4.1 Key Findings and Implications

This paper shows that:

- AI agent access to financial functionality will become a reality rather soon
- While the APIs could be integrated directly into AI agent applications for more explorative and private use cases an MCP based access makes sense
- Service providers should therefore provide both: MCP based access in addition to traditional APIs
- This MCP based approach comes with some functional and technical challenges which need to be addressed
- The functional challenges might make it necessary to re-think the use cases exposed by the current APIs to better suit them for AI agent access

4.2 Recommendations

As outlined in this document, providing AI agent access comes with some new challenges and requirements:

- Providing more context for an AI agent to understand the purpose of an API (an potentially re-think use cases to better suit AI agent access)
- Potentially leveraging the capabilities of the “Back-Channel” (elicitation, sampling) functionality
- Providing capability discovery including fine grained authorization
- Addressing security threats introduced by this new access and interaction pattern

Recommendations for SFTI

As SFTI Common API does not actually implement and operate such AI access but rather defines the interface standards we recommend the following actions:

- Each common API working group should review the current API descriptions and enhance/rework the SFTI common API specifications to include additional metadata already directly within the existing API specifications. This metadata should include additional context information (including examples) and be targeted for AI agents.
- Re-think the use cases supported by the API and elaborate
    - if and how the supported use cases would need to be exposed differently to AI agents via MCP to better guide them
    - if and how the supported use cases could benefit from “Back-Channel” use and whether they need new dedicated endpoints.
- Include capability discovery and extend the common API security concept with fine grained authorization for use of different capabilities. I.E. rework the current consent management definitions and extend the rich authorization data with aspects of capability use.

To do so, we recommend that each SFTI Common API stream should provide simple reference implementations of MCP servers that includes the results of the actions enlisted above. Doing so will make sure that enhancements/extensions are feasible und usable in practice. Implementing a reference MCP server will also allow the working streams to dive into the new interaction patterns introduced by MCP.

Recommendations for Financial Institutions

Agentic access to financial data and functionality will become a near-term reality. Combined with increasing trust in AI applications and the massive speed in technological improvements banks financial institutions will be confronted with some challenges but also opportunities. We recommend that financial institutions ask themselves the following questions:

- How shall we include agentic functionality and access into our channel landscape? E.g. through our own mobile channels or through a third party channel like the mobile OS AI functionality?
- What functionality do we expect to be covered by AI agents? E.g. what kind of advice will be provided by AI agents without human in the loop?
- What functionality (tools) shall we provide for third party AI agents?
- How do we control third party agent access to protect our customers?
- How should agentic access be prized?

4.3 Next Steps

The sounding board recommends the following next steps:

- That SFTI and the financial institutions follow the recommendations
- That it makes sense to translate this white paper into reality. Potentially through a PoC with SFTI Common APIs. Ideally a use case that benefits all members should be chosen.
- That, based on the developments in the field, a new version of the white paper could be published in a year or so

5. Appendix

5.1 Divergent Views of Selected Sounding Board Members

There have been critical views on the MCP approach by a minority of the sounding board members because of the following topics:

- **Insertion of additional layers**. With MCP actually 2 layers will be added. Every layer simplifies something for someone and complicates it for someone else (complexity, delay, maintenance).
- **Missing identification of Agents**: The agent that performs actions on behalf of the user must be identified. This will support autonomous operation of the agent, separation of user and agent actions, easier lifecycle management, traceability and auditability. Agent tokens may be revoked independently. For example, authentication of the agent may be performed based on RFC 8693 (Token exchange). Giving the agent its own identity is the cleanest, most secure way to allow autonomous action on behalf of the user. It solves storage, revocation and audit problems and aligns with OAuth best practices.
- **A „Back Channel“** may introduce the need to implement business logic into the MCP layer (client) because „flow“ may be dictated by the MCP server. This may lead to compatibility issues and/or deep coupling - which is not what initially was intended. May also lead to severe vulnerabilities like progressive extraction of user data and complicates consent and audit challenges.
- **URL mode elicitation** may be implemented for high-security or sensitive user interactions that must not pass through an AI Agent or must happen outside the MCP channel for safety & compliance reasons (e.g., credentials, MFA, payment authorization pages, onboarding, KYC, consents). URL mode elicitation can be used by an MCP server to instruct the MCP client to open an external URL where the user completes the authentication action outside the MCP session. The AI system never receives credentials, PINs, payment details, or regulatory-confirmation data. Instead the credentials are stored only in the MCP server. For further information see https://github.com/modelcontextprotocol/modelcontextprotocol/issues/1036.

5.2 Excursus: Integrating Resources into ChatGPT

OpenAI’s ChatGPT offers “Apps” which allow to integrate resources directly into the conversations. It is primarily a closed ecosystem approach with a growing number of pre-implemented connectors to office tools and business applications. With this approach OpenAI can offer enterprise level control over which connectors (apps) can be used and by whom. But it is not fully closed, it also allows to integrate any third-party system into ChatGPT using MCP – this requires developer mode to be enabled. https://help.openai.com/en/articles/11487775-apps-in-chatgpt

The other possibility is to use “GPT Actions”. They can be stored in Custom GPTs which one can create for specific purposes. GPT Actions can build a direct integration to any REST API endpoint. One needs some developer skills to describe the API schema, configure the authentication and give use instructions so that the Custom GPT can use the API in the conversion. https://platform.openai.com/docs/actions/introduction

5.3 List of References

IBM Think. *What are AI agent protocols?* https://www.ibm.com/think/topics/ai-agent-protocols (Much of chapter 0 is adapted from this article.)

Shanghai Jiao Tong University. *A Survey of AI Agent Protocols*. https://arxiv.org/pdf/2504.16736 (Chapter 0 was enriched with details from this article.)

Anthropic. *About MCP – Architecture overview*. https://modelcontextprotocol.io/docs/learn/architecture (Much of chapter 3.1.4 is adapted from this article.)

RedHat. *Model Context Protocol (MCP): Understanding security risks and controls*. https://www.redhat.com/en/blog/model-context-protocol-mcp-understanding-security-risks-and-controls (Much of chapter 3.2.1 is adapted from this article.)

OWASP Foundation. *OWASP MCP Top 10.* https://owasp.org/www-project-mcp-top-10/ (Chapter 3.2.1 was enriched with details from the list.)

Anthropic. *Security – Understanding Authorization in MCP*. https://modelcontextprotocol.io/docs/tutorials/security/authorization (Much of chapter 3.2.2 3.1.4is adapted from this article.)

Mohamed Azharudeen. *The MCP Privacy Gap: How Model Context Protocol Creates Hidden Data Threats*. https://medium.com/ai-insights-cobet/the-mcp-privacy-gap-how-model-context-protocol-creates-hidden-data-threats-aa802e1b3cf8 (Some of chapter 3.2.3 is adapted from this article.)

ModelContextProtocol.info. *MCP Best Practices: Architecture & Implementation Guide*. https://modelcontextprotocol.info/docs/best-practices/ (Some of chapter 3.3.2 is adapted from this article.)