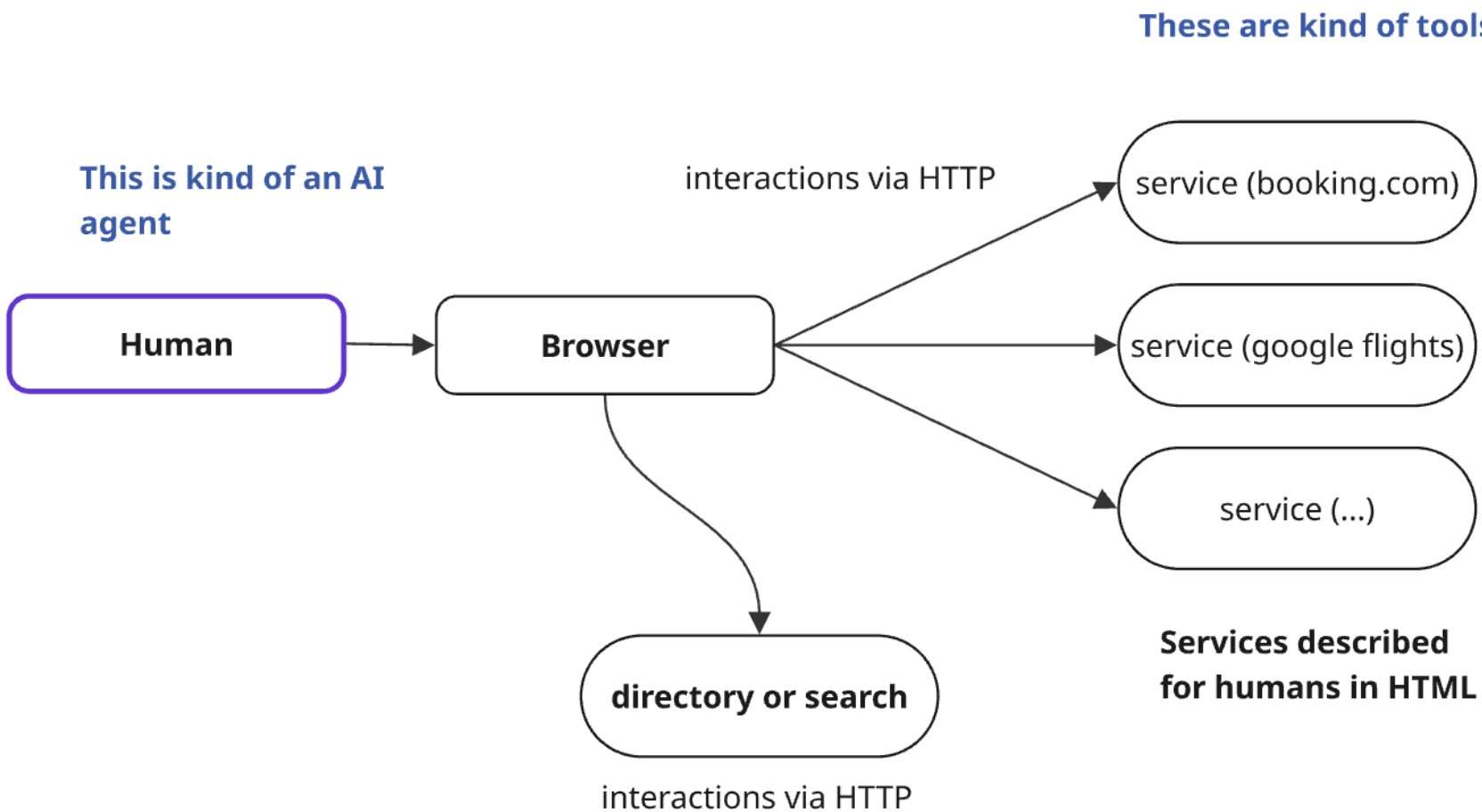
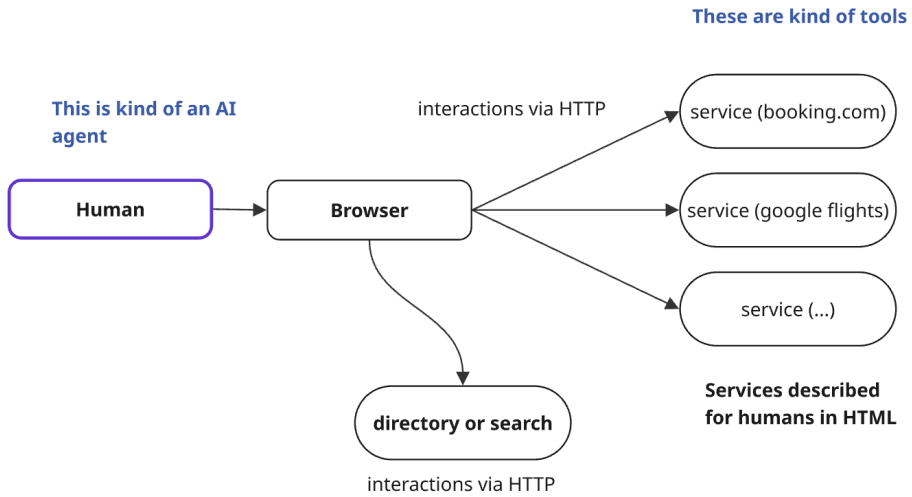
Web for humans: Humans are the "agents"

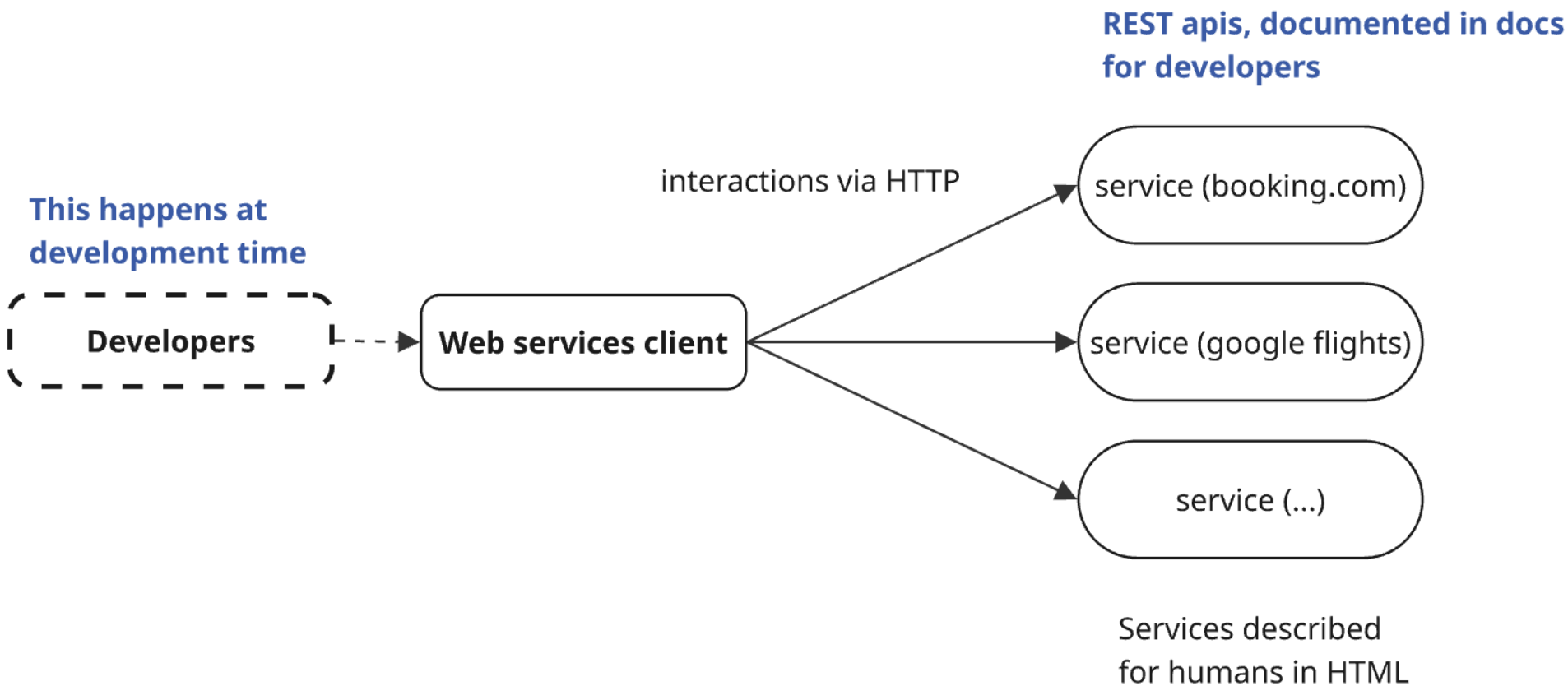


Web for humans: Humans are the "agents"

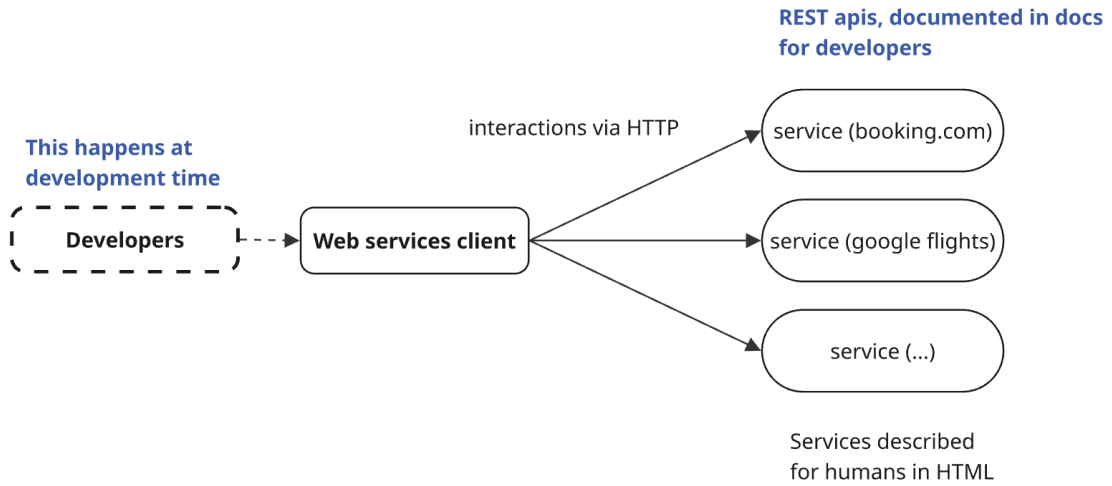


1. Services are described for humans. The language for "services" and "tools" is HTML, rendered as text. Services accessed via HTTP.
2. The description is **comprehensive**, in addition to navigation being **guided**. The web pages drive and guide the interaction through the provider's services. Pages describe the provider, the services and how they fit.
3. Humans ARE the agent, their actions are just "mediated" by the browser. AI has no autonomy. Humans know what they want (kind of) and drive the "browser". No need to describe it. There is **no "autonomy leash"** or "autonomy slider" problem.
4. Upgrades are quite manageable: web pages can change, humans will adapt.
5. Effectiveness is measured via user studies at first, A/B testing, and tools that monitor user journeys. Testing software.
6. Errors and threats come from badly written pages, and (now) fairly well-known web attacks. Software clients are a few, widely-tested browsers
7. Errors and undesirable behaviors are usually easily “visible” /detectable

Web services and SOA

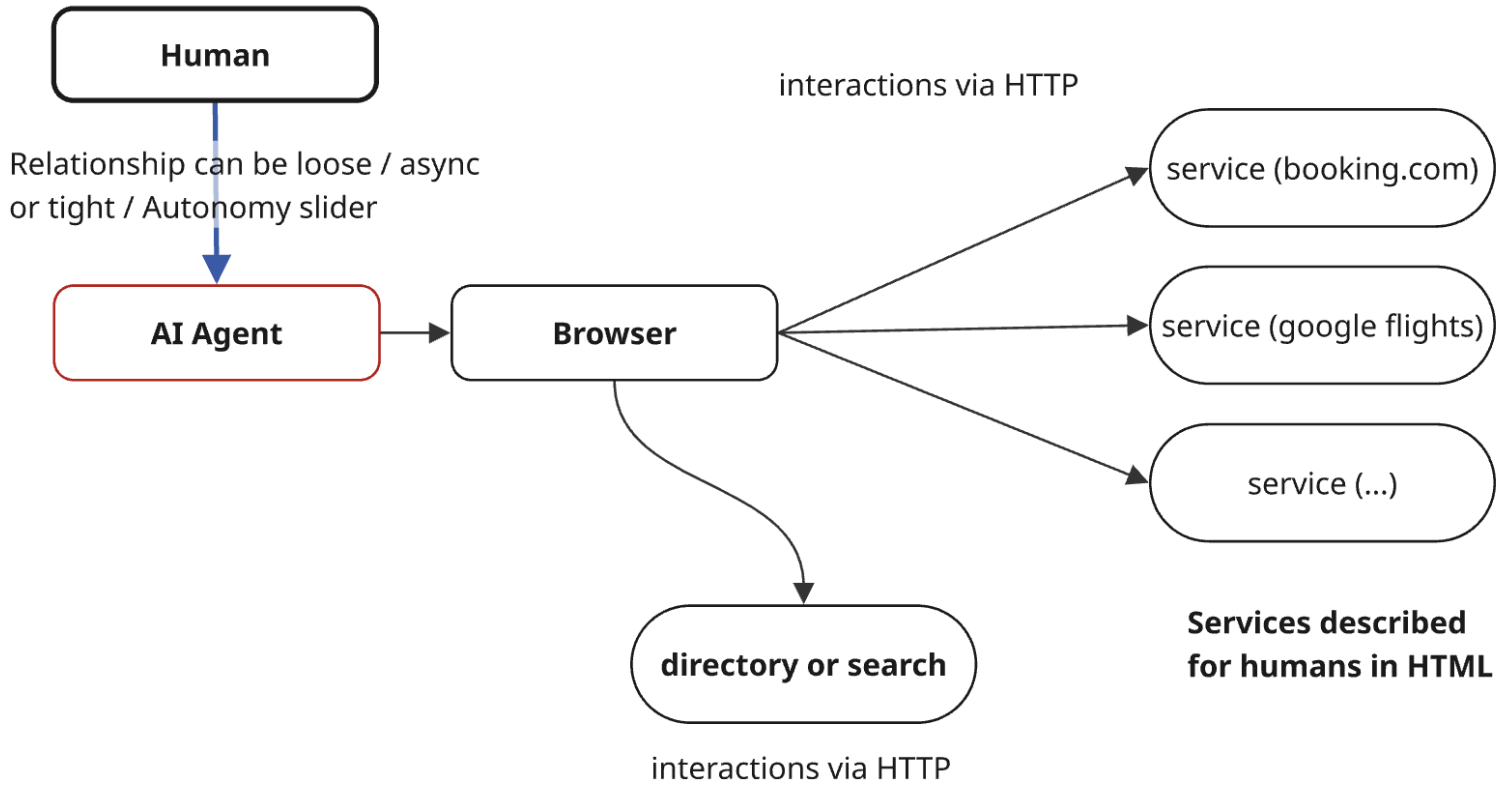


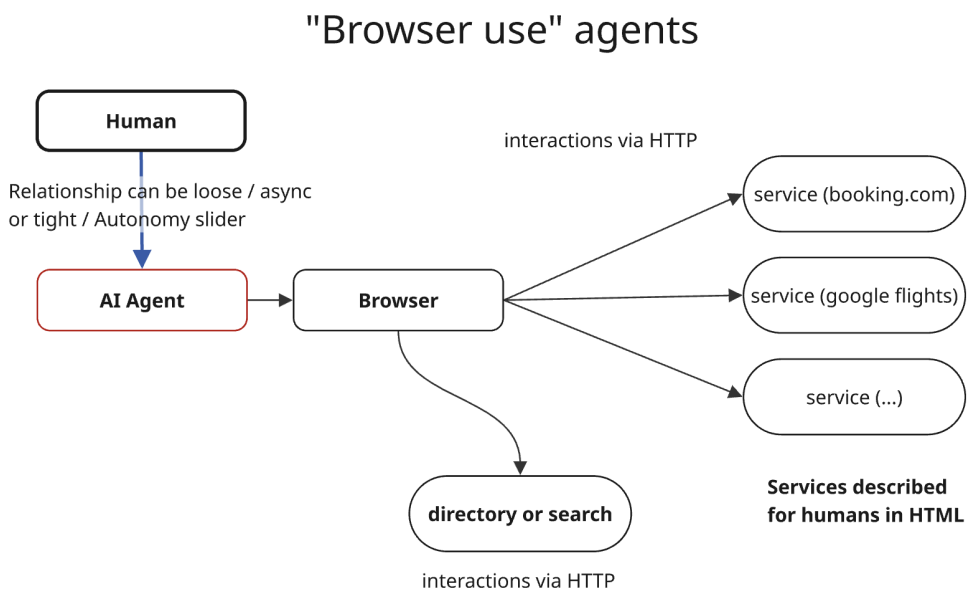
Web services and SOA



1. Services are **described for humans**. The language for specs is English. Interactions take place via HTTP.
2. The description of the service is comprehensive. Navigation is not guided, in most cases - developers of clients should know what they can call when. The correct way to interact is specified in the documentation that developers have to read and understand.
3. Developers code the "agent", a software that has no "intelligence". Either the client is programmed or interacts with humans. The software has **no autonomy**. Humans know what they want (kind of) and the desired behavior needs to be captured by the dev team and coded.
4. Upgrades are tricky, versioning is costly for providers and clients.
5. Testing, logging and monitoring is often basic and meant to assess if code breaks, for the most part.
6. Errors and attack coming from versioning, injections, and the general complexities of distributed systems that have many components and moving parts
7. Errors and undesirable behaviors are usually easily “visible” /detectable, traces are logged and monitored

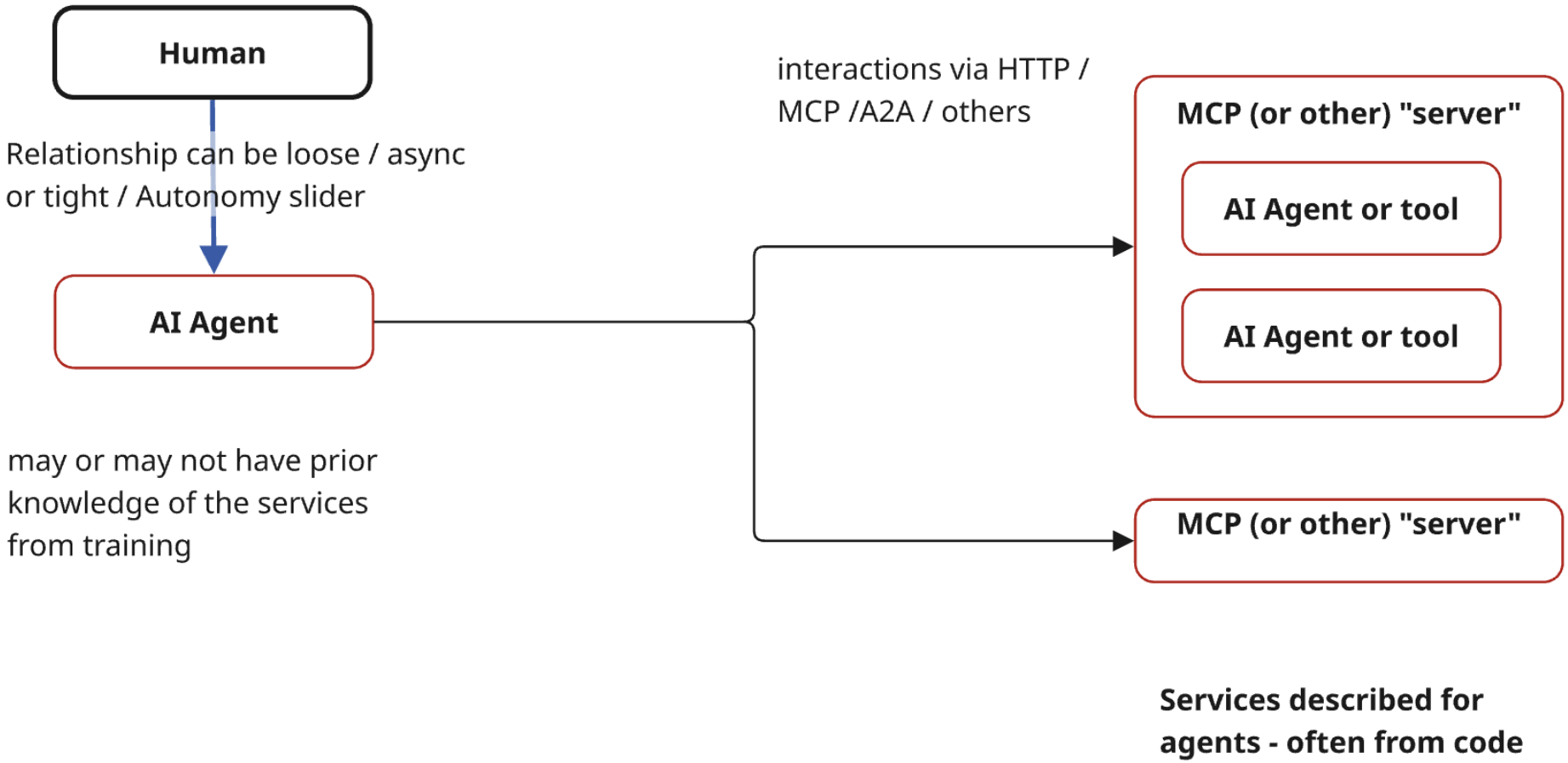
"Browser use" agents



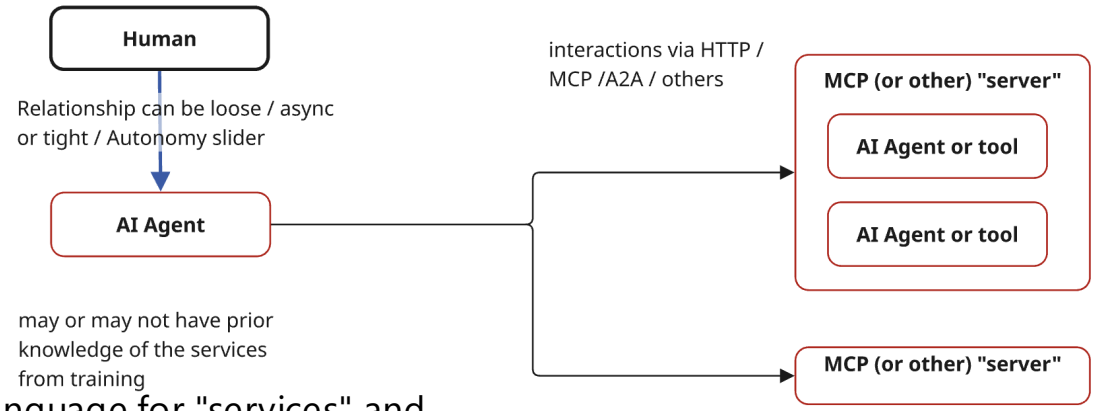


1. Services are described for humans. The language for "services" and "tools" is HTML, rendered as text. Services accessed via HTTP.
2. The description is **comprehensive**, in addition to navigation being **guided**. The web pages drive and guide the interaction through the provider's services. Pages describe the provider, the services, and how they fit.
3. **"Agency" is mixed** - and sits on a scale, depending on how much leash the agent is given. This model is still evolving and ranges from AI only providing a summary view of a page to AI creating plans and confirming them, to AI running off doing what they think makes sense.
4. Upgrades are quite manageable: web pages can change, agents will adapt.
5. Testing, logging and monitoring is still developing. No clear pattern emerged yet - except for "advanced" agents logging and learning from the past
6. Errors may come from these additional sources: a. agents not understanding content, b. wrong degree of autonomy, c. attacks - still to be fully understood
7. It's hard to say how manifest errors and undesirable behaviors are, but for now this approach has a mostly short leash so they are fairly visible

Agents in agentic ecosystem - tools



Agents in agentic ecosystem - tools



1. Services are described for agents - kind of, and, whatever that means. The language for "services" and "tools" is JSON, often taken from python specs. Services accessed via HTTP/JSON-RPC.
2. The description is spotty, no best practice emerged. **Navigation is not guided**. In web pages I have the page structure and access to the whole web site to read. What is the equivalent here?
3. **"Agency" is mixed** - and sits on a scale, depending on how much leash the agent is given. Notions of memory and persistence of user knowledge are also emerging.
4. Changes / Upgrades are in principle manageable if agents are well developed and manage to combine training knowledge, prior execution knowledge, and updated service description. Service descriptions change and agents can adapt.
5. Testing, logging and monitoring is still emerging - and the notion of traces and "actually useful process mining" are emerging. Approaches to automate agent definition analysis have also emerged.
6. Errors may come from i. agents not understanding content (and service descriptions not providing sufficient context), ii. wrong degree or autonomy, iii. attacks - still to be fully understood, iv) misunderstanding on what/how much the agent knows by training vs what is described
7. Agents at scale - the "software engineering" moment: what happens as millions of people with little sw eng experience develops complex multi entity systems. (see next)
8. In many implementations, error are suppressed -> see later again for the “software engineering” crisis. Legal framework somewhat unclear

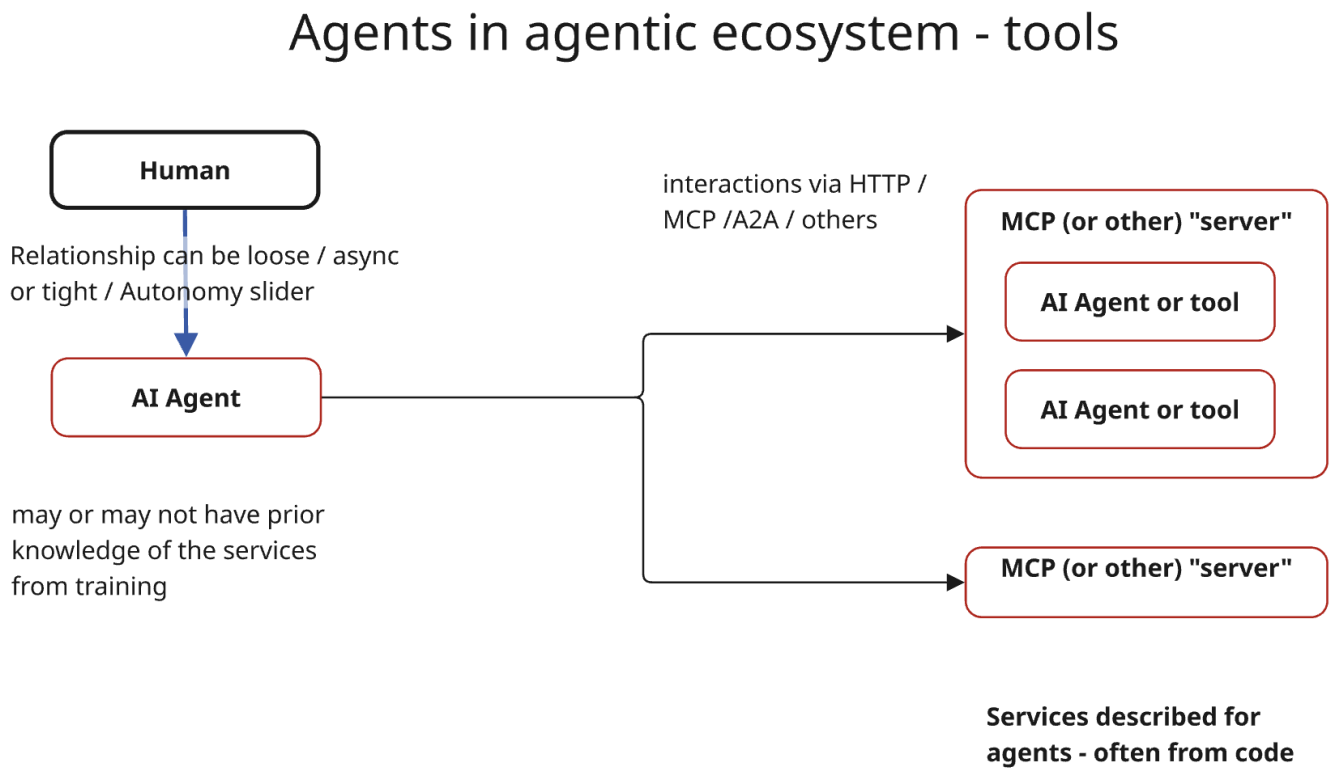
DESIGN CONSIDERATIONS FOR AI AGENTS

(as a testament to the maturity of the field, this is what the powerpoint designer came up with)

Please consider each of them rfom the point of view of the vaiorus parties involved: who should worry about that design consideration?

AI agents – systems of LLM and tools - are complex distributed systems

- They have all the complexities of distributed systems, minus one, plus many more
- Ease of development does not mean that their reliable operation is easier



AI agents – systems of LLM and tools - are complex distributed systems, and….

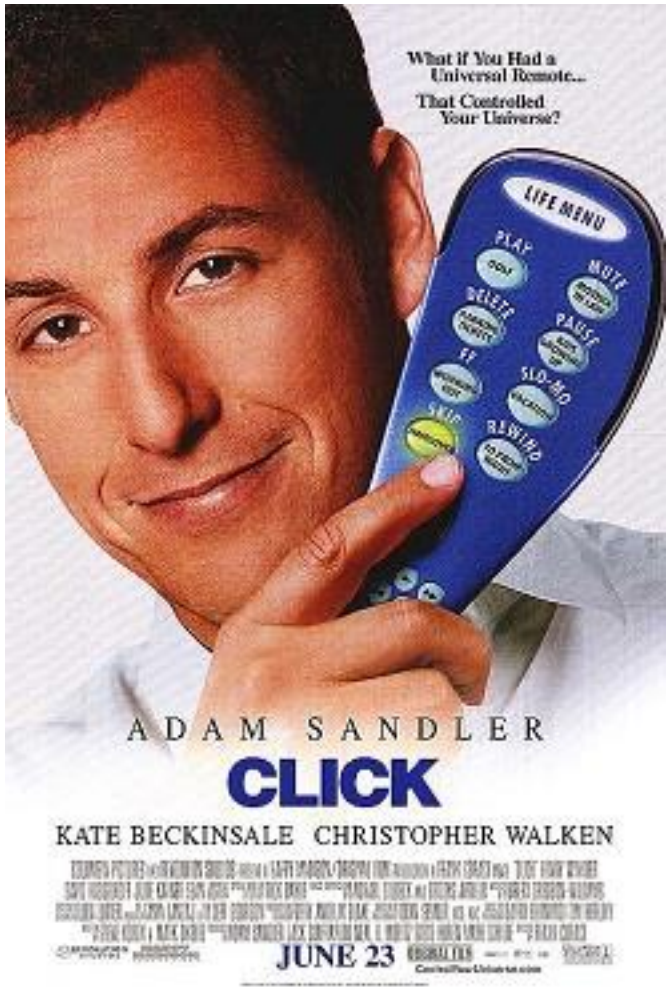
- …we currently have millions of new such systems developed daily – the overwhelming majority of which by people who have no idea what software is, or what engineering is.
- These people are using **our** tools to build systems.
- If we expose an MCP, either it is not used (not what we want) or it is used (also not what we want)

Memory, Intentions, Autonomy

- Short term memory: how to “design” it, and who contributes or suggests content?
- And long term / cross session memory?
- How do we elicit preferences and intentions and delimit their scope?
- How much should be inferred vs asked?
- Degree of **autonomy is a dimension and intersects with memory**
  - Declarative goals? - and our level of comfort

⇒What’s the role of the MCP server? (For example, Render is wisely very explicit)

⇒How much should we expose and to who? How to ask? How to show? How often?



Security, safety, privacy

- Here I have nothing particularly clever to say that you don't know already
- AI systems are by definition handling nuanced and non trivial cases
- They may make all sorts of wrong or "unfair" decisions
- They may be led to do the wrong thing, and this can happen from so many sources once our client is connected to a bunch of servers - to which we probably lost track of

- The issue arises both for providers and consumers of both intelligent and dumb services, and the responsibility is shared. We need to ask ourself the questions, no matter our role.
- Logs of AI interactions: can admins see what people share? What they… think?

Guided interactions

- how can service providers give an integrated overview of what they offer (rather than just api by api).
- How can interaction and “next steps” be guided, or should they?
- On the web, both humans and developers do have context.
  - Humans or agent browsing web sites have context. what about agents with mcp-style descriptions?
  - how to be intentional on what we expose?

- Question for us: how much guidance to give? How open vs prescriptive should our description and answers be? How careful on what we expose?
- How can we foresee “bad usage” of our “api”? How to prevent it?

Quality, Logs, Traces, Business assertions /1

- As both provider and clients we want to know
  - How are services being used? Which are the common patterns? Where do clients drop off?
  - ➔ very similar to how we analyze users on the web
- In addition, and this is new, we know have millions of “new software engineers” with wildly differing practices on how to manage exceptions
  - Our job is to be robust to incompetent users
- Key elements to this are **telemetry for traces** and the notion of **business assertion.** They try to fight the problem of **silent failures**
  - what are we really monitoring for? How to do it at scale?
  - Often – there are properties of an interaction that we expect to be true **individually** and properties that should be true **statistically**

Quality, Logs, Traces, Business assertions /2

- Log “consent” when possible (and what/ who consents to what)
- Automated error analysis and improvement. This now possible. In fact **we are in a golden era: this is EASY today**.
  - The easiest consulting money you will ever make, revenue of around 50 K / day easy
  - Services today are so bad that even models by [omitted on advice from legal] will detect them

  In summary we need to think and implement abstractions and practices to manage complex, semi-autonomous system and describing good vs bad behaviors.
- Self improvement is also kind of easy– needs to be on short leash

Capturing opportunities

- Ability to
  1. read docs at scale, understand complex systems
  2. answer direct questions about "how to do x”
  3. interact conversationally and create custom UI
- Responsibilities
  - in the web we have terms and conditions - how does it work here?
  - who is responsible for actions? does it depend on how clear the service descriptions are?