Creating Your Own MCP-Style Agentic System Right Now

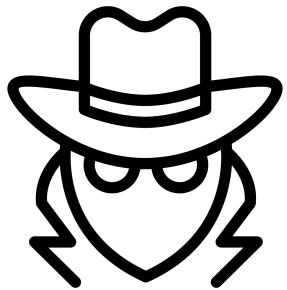
MikeAmundsen
@mamund


Mike Amundsen
@mamund

Overview

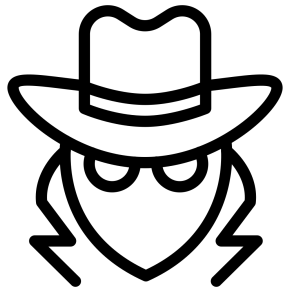
- Agentic Essentials
- Discovery
- Composable Services
- Shared State
- Job Control
- Parting Thoughts

Agentic Essentials



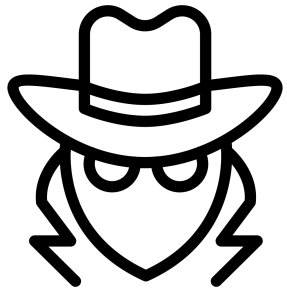
The Rise of Agentic Systems

- What do we mean by “agentic”?
- Why is it popular now?
- What infrastructure is required?



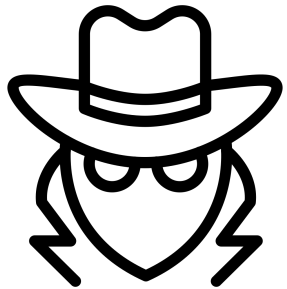
Goal-Driven, Autonomy-Friendly Systems

- Agent: an app that pursues goals
- Agentic: acting as an autonomous agent
- Often mentioned for AI-driven tasks
- But not limited to LLM world



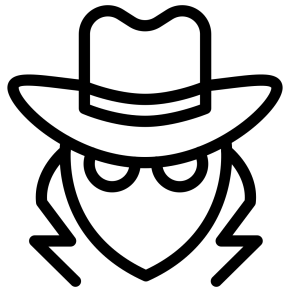
The Agent Moment Has Arrived

- LLMs unlocked new possibilities
- MCP (Message-Centric Protocol) makes APIs agent-friendly
- Tools like LangChain, CrewAI, AutoGen, MetaGPT are popular



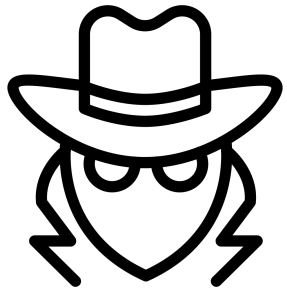
Agentic ≠ Autonomous AI Magic

- Agentic systems rely on structure and constraints
- The agent is only as good as the environment it acts within
- Real-world systems must be predictable, safe, reversible



Three Pillars of Agentic Systems

- Composable Interfaces: clean, stable APIs with consistent affordances
- Shared State: memory across steps and services
- Service Discovery: agents need to find and understand what's available



Let’s see an agent action …

My job-runner agent

```
mca@mamund-ws:~/.../scripts$ job-runner --help
Usage: job-runner [options] <jobFile>

Run a job-control document with optional state handling

Arguments:
  jobFile              Path to the job-control JSON file

Options:
  --state <file>       Initial shared state JSON file
  --job-url <url>      Job-control base URL
  --state-url <url>    Shared-state base URL
  --emit [file]        Write final shared state to file or stdout if no file is given
  --keep               Retain shared state after job ends
  --overwrite          Overwrite existing shared state if ID already exists
  --dry-run            Print but don't execute HTTP requests
  --verbose            Print debug information
  -h, --help           display help for command
mca@mamund-ws:~/.../scripts$
```

My job-runner agent

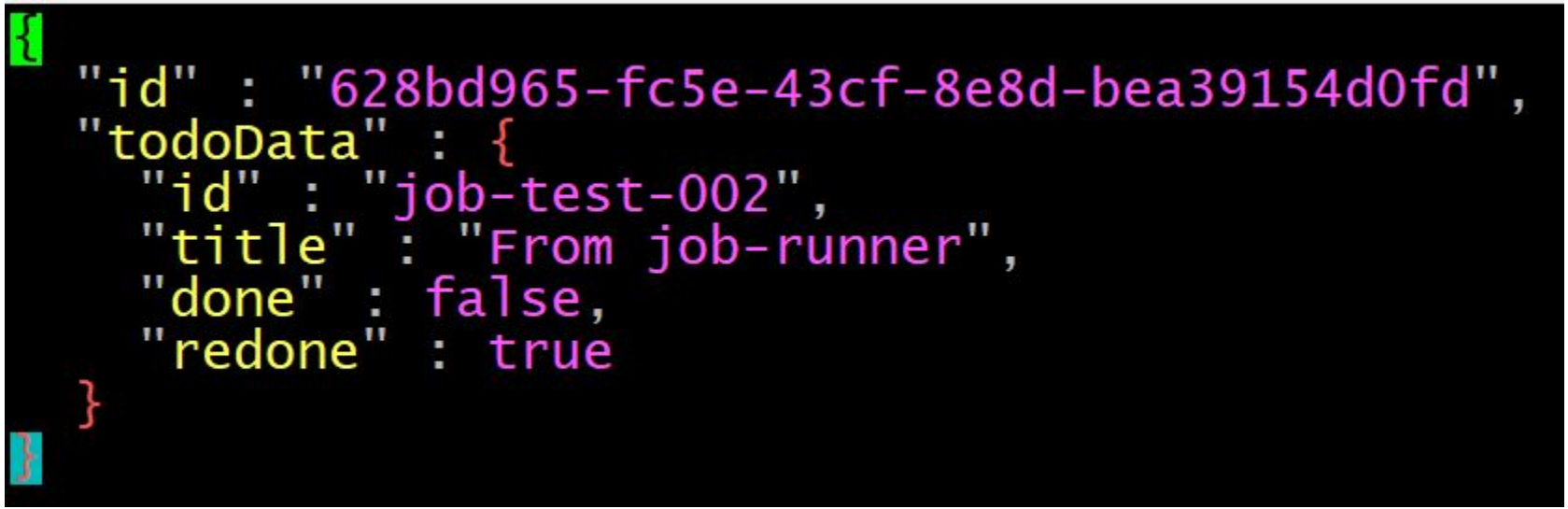
- Take instructions from a jobFile
- Maintain state values between API calls
- Produce output for review

Todo JobFile

```
{
  "sharedStateURL": "http://localhost:4500/state/628bd965-fc5e-43cf-8e8d-bea39154d0fd",
  "name": "test-todo-service",
  "description" : "A short script to exercise the TODO service",
  "steps": [
    {
      "name": "delete-todo",
      "description" : "to start w/ a clean slate, delete anything from the last run",
      "enabled": true,
      "tasks": [
        {
          "tag": "todo",
          "serviceName" : "todo-service",
          "enabled" : true,
          "input": {
            "command": "delete",
            "resource": "todo",
            "id": {"$fromState" : "/todoData/id"}
          },
          "storeResultAt": [
            {
              "targetPath": "/state/todoDelete"
            }
          ]
        }
      ]
    },
    {
      "name": "create-todo",
      "enabled": true,
      "tasks": [
        {
          "tag": "todo",
          "serviceName": "todo-service",
          "input": {
            "command": "create",
```

My job-runner memory

- The agent will need to know what data to deal with
- We can supply initial values via a state document



My job-runner

```
#!/bin/bash

# run a job

job-runner todo-test.json \
     --state todo-test-state.json  \
     --overwrite \
     --emit \
     --keep | jq .

# EOF
~
```

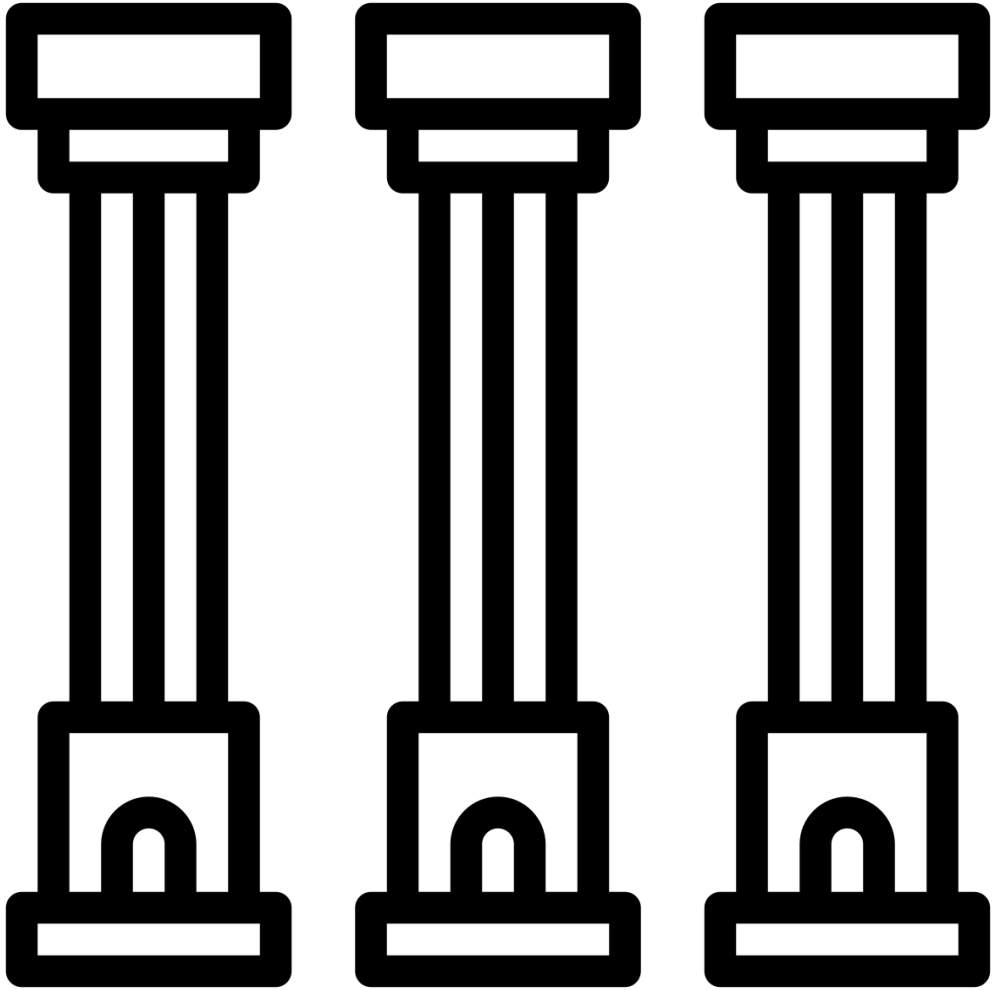
Resulting output

What did we just see?

- Define a job
- Set starting memory values
- Tell the agent to run the job
- Check results

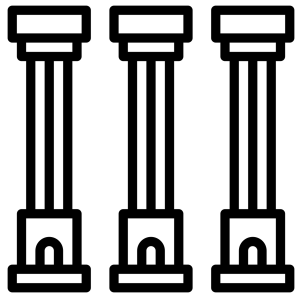
What makes this possible?

- Job control gives us structure
- But the real power lies deeper
- Agentic systems depend on flexibility and coordination
- That’s where the Three Pillars come in



Three Pillars of Agentic Systems

- **Discovery**
  - **Find capabilities at runtime**
- Composable Services
  - Expose common interface
- Shared Memory
  - Act as “glue” across API calls



Discovery

Finding Capabilities

- Support “Find and Bind”
- Based on OPEN-DISCO
- Providers register a capability
- Consumers search for capabilities

Open DISCO

Summary

This document details an *adhoc specification* called **DISCO** (**D**iscovering **I**nteroperative **S**ervices for **C**ontinuous **O**peration). DISCO is a simple language for managing the adding/removing of services as well as the ability to search ("find") and make connections with ("bind") registered services.

DISCO was designed to be easy, open, lightweight, and extensible. For this reason, readers/implementers may find things "missing" or "underspecified." This is intentional. Getting started is meant to be easy. And local customization is supported as needed. This allows the DISCO spec to safely grow and improve over time without breaking exsting implementations.

The DISCO "language" supports the following features:

- `register` : add a service to the shared registry
- `find` : query the registry for services (dependents) to consume
- `bind` : notify the registry the intention to connect with and use another service
- `renew` : renew a service's registry *lease* to prove is is still up and running
- `unregister` : remove a service from the registry

http://open-disco.org/

Providers Register

````
#### Example with `curl`
```bash
curl -X POST http://localhost:4000/register \
  -H "Content-Type: application/json" \
  -d '{
    "serviceName": "example",
    "serviceURL": "http://localhost:6000",
    "tags": ["example"],
    "pingURL": "http://localhost:6000/ping"
  }'
```
````

Providers Register

- Name
- URL
- Search tags
- Optional ping URL

Consumers Search

````
### `GET /find?tag=todo`
Finds services registered under a given tag.

#### Response
```json
[
  {
    "serviceName": "todo-service",
    "serviceURL": "http://localhost:5000",
    "tags": ["todo", "task"]
  }
]
```
````

Consumers Search

- Pass tag(s)
- Select service from list
  - Via URL
- Optionally, check ping for status

Demo


{"timestamp":"2025-09-30T10:40:36.981Z","level":"info","action":"startup","port":4000}
{"timestamp":"2025-09-30T10:40:38.028Z","level":"info","action":"register","registryID":"601332e7-b150-4c00-9ba4-6c33f5f9e600","serviceName":"shared-state","serviceURL":"http://localhost:4500"}
{"timestamp":"2025-09-30T10:40:40.139Z","level":"info","action":"register","registryID":"93915ef1-9090-48ff-a720-19ac8e012c5c","serviceName":"todo-service","serviceURL":"http://localhost:4001"}
{"timestamp":"2025-09-30T10:40:40.172Z","level":"info","action":"register","registryID":"b8196883-373c-4a32-8204-fe1b02184d0f","serviceName":"service-b","serviceURL":"http://localhost:4200"}
{"timestamp":"2025-09-30T10:40:40.213Z","level":"info","action":"register","registryID":"fbe5f7e7-f6cc-4648-a680-e9f41a8ae40d","serviceName":"service-a","serviceURL":"http://localhost:4100"}
{"timestamp":"2025-09-30T10:40:40.235Z","level":"info","action":"register","registryID":"b7aeb342-888c-4337-a9b9-b4593ce23107","serviceName":"person","serviceURL":"http://localhost:4600"}
{"timestamp":"2025-09-30T10:40:40.240Z","level":"info","action":"register","registryID":"06dbf9dd-eab9-4ad8-9fb9-653bc76d897c","serviceName":"service-c-writer","serviceURL":"http://localhost:4300"}
{"timestamp":"2025-09-30T10:40:40.252Z","level":"info","action":"register","registryID":"c44952dc-944b-4abd-b932-e17841c4167f","serviceName":"service-c-reader","serviceURL":"http://localhost:4400"}
{"timestamp":"2025-09-30T10:40:40.253Z","level":"info","action":"register","registryID":"10bb7ffc-c160-477a-8aa9-1a9df1bf8db2","serviceName":"openweather","serviceURL":"http://localhost:4602"}
{"timestamp":"2025-09-30T10:40:59.545Z","level":"info","action":"find","filters":{"tag":"todo"},"matches":1}
{"timestamp":"2025-09-30T10:40:59.589Z","level":"info","action":"find","filters":{"tag":"todo"},"matches":1}
{"timestamp":"2025-09-30T10:40:59.606Z","level":"info","action":"find","filters":{"tag":"todo"},"matches":1}
{"timestamp":"2025-09-30T10:40:59.618Z","level":"info","action":"find","filters":{"tag":"todo"},"matches":1}

Provider registers on startup

```
// Self-registration with discovery
async function registerWithDiscovery() {
  try {
    const registryURL = 'http://localhost:4000/register';

    const serviceInfo = {
      serviceName: 'todo-service',
      serviceURL: `http://localhost:${PORT}`,
      tags: ['todo', 'create', 'read', 'update', 'delete', 'filter'],
      semanticProfile: 'urn:example:todo',
      mediaTypes: ['application/json'],
      pingURL: `http://localhost:${PORT}/ping`
    };

    const response = await axios.post(registryURL, serviceInfo);
    log('register', { registryID: response.data.registryID });
  } catch (err) {
    log('register-failed', { error: err.message }, 'error');
  }
}
```

Consumer finds and binds

```
const response = await axios.get(discoveryURL, {
    params: { tag: task.tag }
  });

  if (!response.data.length) throw new Error(`No service found for tag ${task.tag}`);
  const service = response.data[0];
  const form = await axios.get(`${service.serviceURL}/forms`);

  const mode = step.mode || 'execute';
  const targetForm = form.data.find(f => f.rel === mode);
  if (!targetForm) {
    //throw new Error(`No form found for mode '${mode}'`);
    log('step-unknown-mode', { step: step.name, mode }, 'warn');
    return { status: 'skipped', reason: 'unknown mode', step: step.name };
  }
```

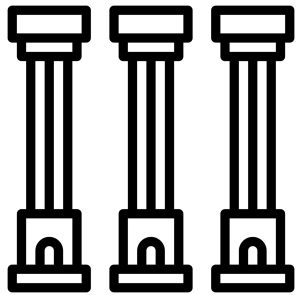
Discovery Solved

- Services register at startup
- Clients search the registry at runtime
- Matched searches results in “Find and Bind”
- Services unregister at shutdown

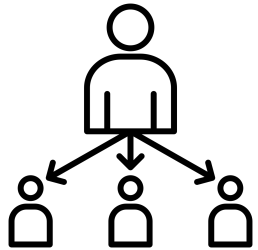
https://mamund.substack.com/p/from-discovery-to-execution

Three Pillars of Agentic Systems

- Discovery
  - Find capabilities at runtime
- **Composable Services**
  - **Expose common interface**
- Shared Memory
  - Act as “glue” across API calls

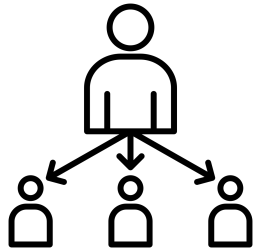


Composable Services



Why Composable?

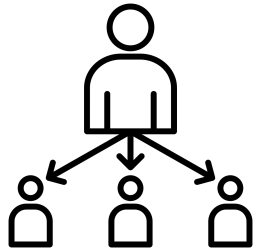
- Agents need a stable interface
  - Same for any capability
- Agents need consistency in input/output
  - Pass structured msgs, not data object
- Agents need repeatable and reversible actions



Three Universal Actions

- **`execute`**: perform the core operation
- **`repeat`**: idempotent re-run or state fetch
- **`revert`**: undo the last operation

These three affordances enable state-aware, reversible workflows



Three Universal Actions

```
app.post('/execute', async (req, res) => {
  const { name, args } = req.body;
  if (name !== 'getWeather') return res.status(400).json({ error: 'Unsupported command' });

  try {
    const result = await getWeather(args);
    log('execute-success', { name, args, result });
    res.json(result);
  } catch (err) {
    log('execute-error', { error: err.message }, 'error');
    res.status(500).json({ error: err.message });
  }
});

app.post('/repeat', (req, res) => {
  try {
    const result = repeatLast();
    log('repeat-success', { result });
    res.json(result);
  } catch (err) {
    log('repeat-error', { error: err.message }, 'error');
    res.status(400).json({ error: err.message });
  }
});

app.post('/revert', (req, res) => {
  const result = revert();
  log('revert-complete', { result });
  res.json(result);
});
```

Passing Structured Messages

```
echo "2. Create a TODO with ID 'test-009'"
curl -s -X POST http://localhost:4001/execute \
  -H "Content-Type: application/json" \
  -d '{
    "command": "create",
    "resource": "todo",
    "id": "test-009",
    "payload": {
      "title": "Walk the dog",
      "done": false
    }
  }' | jq .
echo -e "\n"
```

Designing a Service Interface Wrapper

```
{
  "title": "OpenWeather Wrapper",
  "version": "1.0.0",
  "description": "Composable wrapper for querying current weather from
  "serviceInfo": {
    "serviceName": "openweather",
    "description": "Provides weather data via the OpenWeather API",
    "tags": ["weather", "proxy", "external"],
    "mediaTypes": ["application/json"]
  },
  "resourceType": "weather",
  "resourceSchema": {
    "weather": {
      "temperature": { "type": "number" },
      "conditions": { "type": "string" }
    }
  },
  "authorization": {
    "roles": []
  },
  "commands": {
    "getWeather": {
      "description": "Gets current weather by city",
      "transitionType": {
        "safe": true,
```

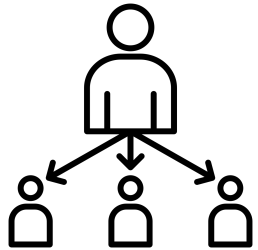
We an even wrap other APIs

```
mca@mamund-ws:~/.../jobs$ job-runner open-weather.json
{
  "title": "open-weather api",
  "input": {
    "weather": {
      "city": "Buenos Aires",
      "units": "metric"
    }
  },
  "output": {
    "weather": {
      "temperature": 20.26,
      "conditions": "Clear"
    }
  }
}
```

Composable Services

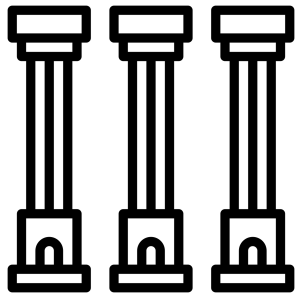
- Three Universal Actions
  - `execute, repeat, revert`
- `design.json` for exposing interfaces
- Wrapping existing APIs for composability
  - open-weather API

Agents can now compose solutions w/ capabilities

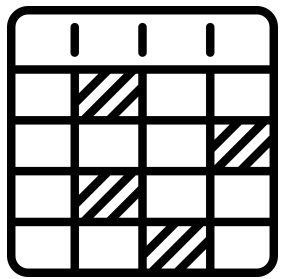


Three Pillars of Agentic Systems

- Discovery
  - Find capabilities at runtime
- Composable Services
  - Expose common interface
- **Shared Memory**
  - **Act as “glue” across API calls**

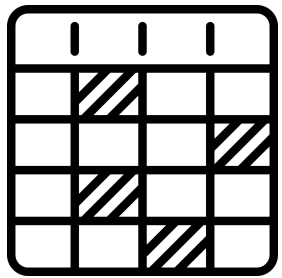


Shared State



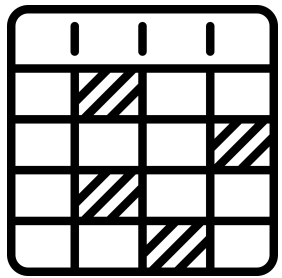
Shared State: The Missing Piece

- Composable, findable services are fine
- Agents that can execute operations are good
- But what do we do with the **results** of API calls?



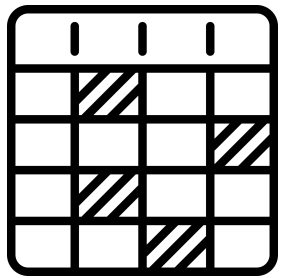
Shared State: What We Need

- We need a shared space to store/retrieve data
- External to any agent or service
- Supports read/write during execution of workflows



Coordination Without Coupling

- Services remain stateless
  - No private internal session needed
- Agents and jobs can pass data without modifying APIs
  - The API doesn’t need to know
- Enables multi-step, multi-service, multi-agent coordination



A simple HTTP interface

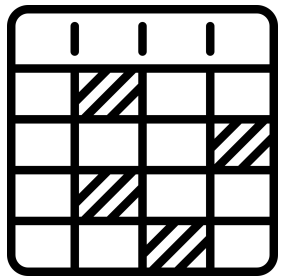
```
### `POST /state`
Create a new shared state document.
- **Body:** `{ id?: string, key: value, ... }`
- **Response:** `{ stateURL: string }`

### `GET /state/:id`
Get document contents (no metadata).
- **Returns:** Raw JSON
- **No metadata is included**
- **No link to metadata is provided (for security)**

### `GET /state/:id?meta`
Get only metadata for the document.
- **Returns:** `{ id, createdAt, lastModified }`

### `POST /state/:id`
Merge new values into an existing document.
- **Body:** `{ key: value, ... }`

### `PATCH /state/:id`
Patch an existing document using:
- `{ op: "add", path: "/some/path", value: ... }`
- `{ op: "merge", value: { ... } }`

### `DELETE /state/:id`
Remove a document. No error if it does not exist.

### `GET /state`
List metadata for all known state documents.
- Each entry includes: `{ id, createdAt, lastModified, rel: "item", href }`
- **Does not include document content**
```

State documents are the universal memory

```
mca@mamund-ws:~/.../scripts$ ./test-shared-state-merge.sh
Creating shared state document...
{
  "error": "State ID 'test-123' already exists."
}
Merging new values into shared state...
{
  "status": "merged",
  "state": {
    "id": "test-123",
    "content": {
      "foo": "original",
      "bar": 42,
      "baz": "new!"
    },
    "createdAt": "2025-09-30T13:28:23.685Z",
    "lastModified": "2025-09-30T13:37:40.986Z"
  }
}
Sending invalid merge value (not an object)...
{
  "error": "Invalid merge value"
}
Sending unsupported operation (replace)
```

Shared State Summary

- Shared state decouples logic from services
- The glue between autonomous steps
- Enables resilient, auditable, and agent-friendly jobs



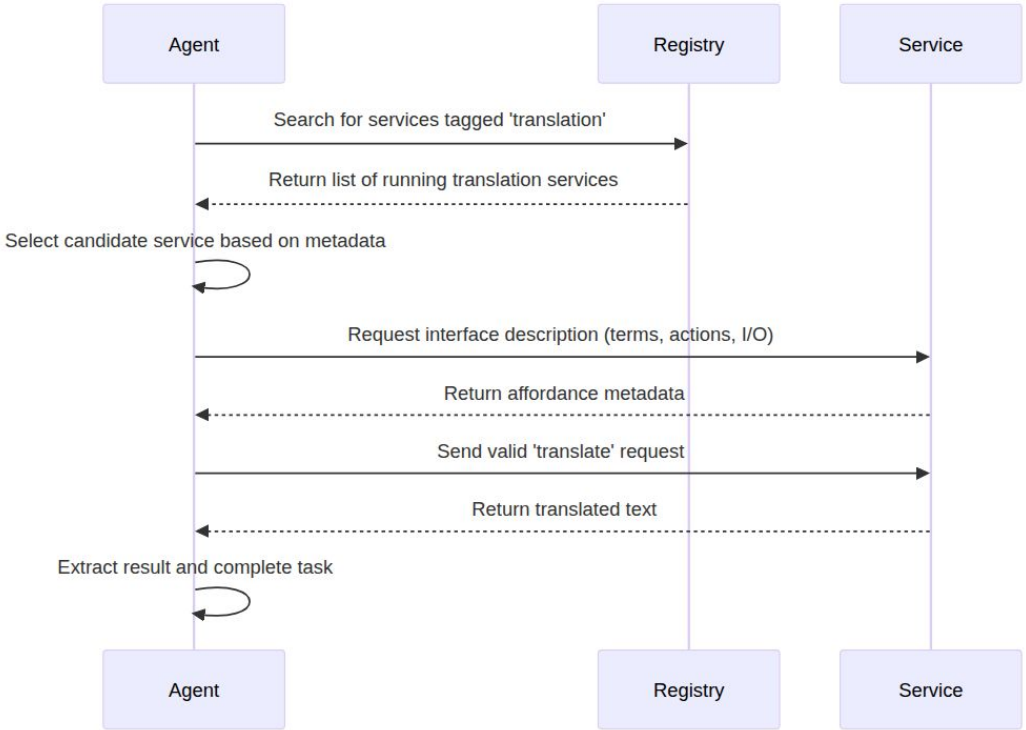
Pulling it all together

The To-Do Agentic Loop

- Composable To-Do service
- Agentic job-runner to process data
- Discovery to match services to agents
- Shared state to handle memory

```
7. List all TODOs
[
  {
    "id": "job-test-002",
    "title": "From job-runner",
    "done": true
  },
  {
    "id": "lw4mhkg3",
    "title": "Buy milk",
    "done": false
  },
  {
    "id": "y95as7ul",
    "title": "Buy milk",
    "done": false
  },
  {
    "id": "test-009",
    "title": "Walk the dog",
    "done": true
  }
]
```

Full lifecycle of agentic operations



And So …

Agentic Systems

- Agents can be autonomous and still be deterministic
- You don’t need LLMs to start today
- You do need an agent platform
  - Discovery
  - Composability
  - Shared state

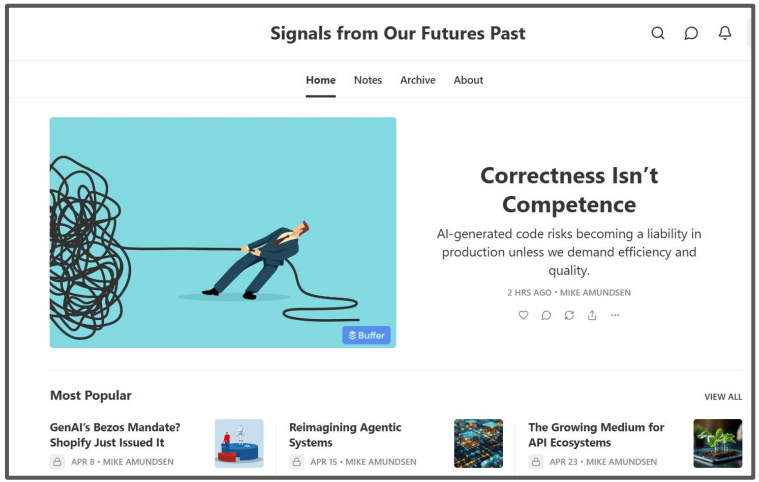
Three Pillars

- Discovery matches service capabilities w agents
- Composability makes enlisting services easy
- Shared state offers the glue
  - Multi-step
  - Multi-service
  - Multi-agent

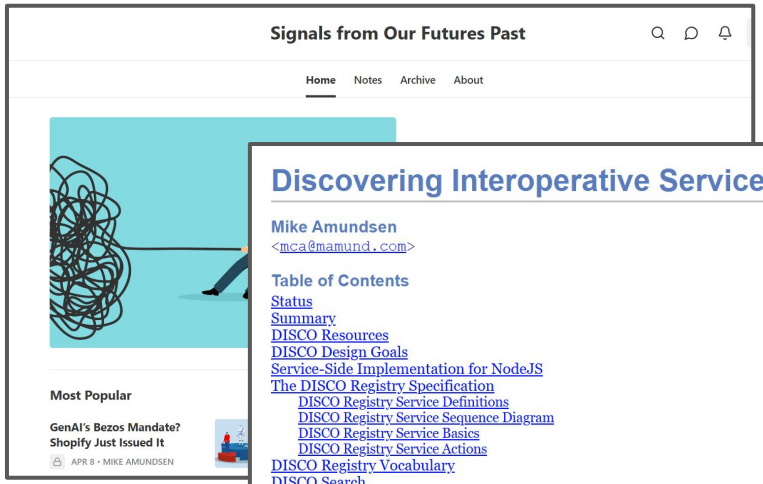
Call to Action

- You have all the parts of an agentic platform already
- Start by building composable service interfaces
- Add discoverability
- Establish shared-state
- Finally, create job-runner agents

Additional Resources



Additional Resources

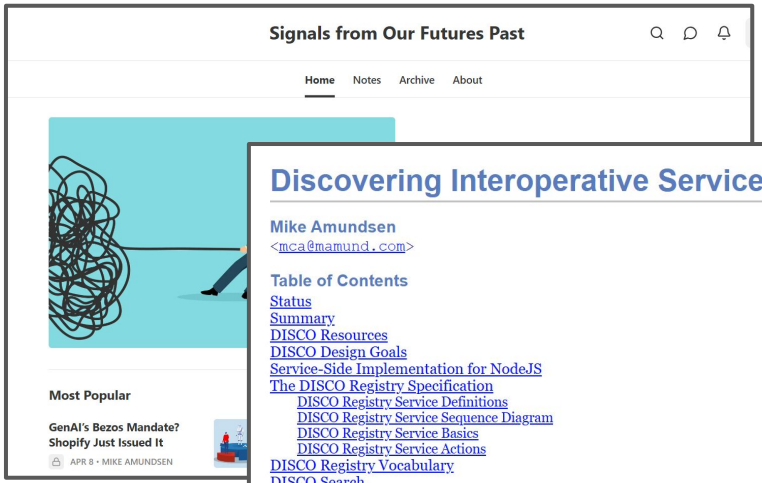


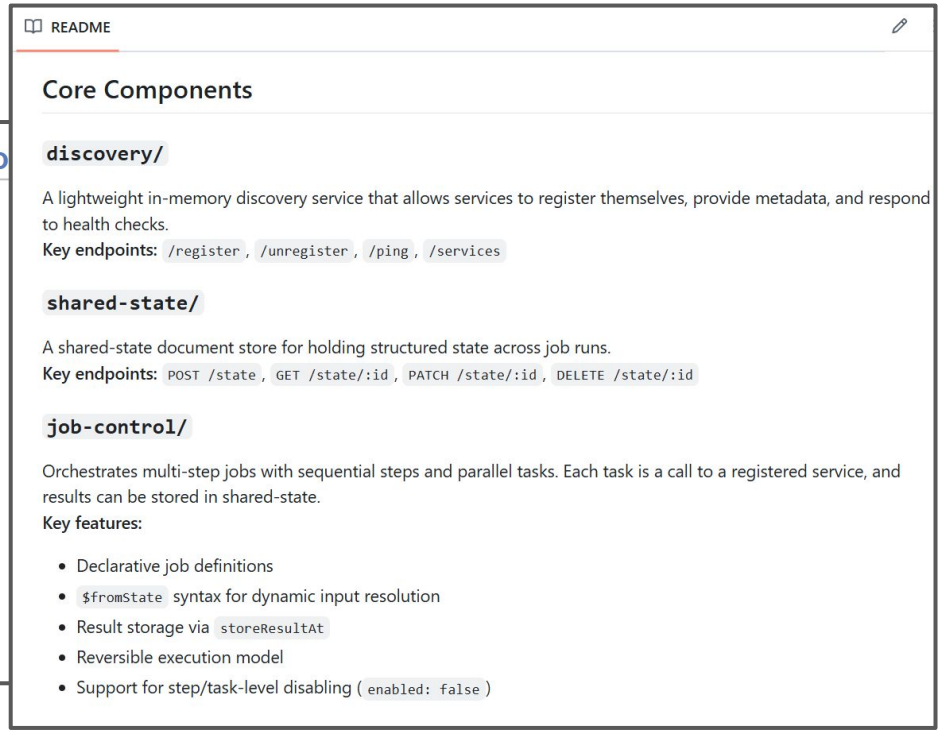
Discovering Interoperative Services for Continuous Operation (DISCO)

Mike Amundsen
<mca@mamund.com>

Table of Contents

Additional Resources





"The best way to predict the future is to create it."

-- Alan Kay

Alan Kay: computer scientist, pioneer of object-oriented programming and the personal computer.

Creating Your Own MCP-Style Agentic System Right Now

MikeAmundsen
@mamund