Agentic AI Service Architecture Based on SOA

Dong Bin Choi[1], Yunhee Kang[2]
and Young B. Park[3,*]

[1]*Department of Computer Science, Dankook University, Korea*
[2]*Department of Computer Science, Baekseok University, Korea*
[3]*Department of Software, Dankook University, Korea*
E-mail: dbchoi85@gmail.com; yhkang@bu.ac.kr; ybpark@dankook.ac.kr
[*]*Corresponding Author*

Received 30 September 2025; Accepted 27 November 2025

Abstract

Service-Oriented Architecture (SOA) structures applications into collections of modular, independent, and reusable services. We propose an SOA-based intelligent service agent framework for building AI applications that decomposes complex tasks into independent functional units. In the framework, the agent operates as an intelligent executor that dynamically orchestrates and invokes diverse services and tools to achieve its goals. The agent is exposed as a self-contained service with a well-defined API, allowing external applications to invoke it directly. By instrumenting requests and responses at both the service and agent layers, the framework enables tracing of the agent's capabilities, performance, and decision-making. We present the design of an operational scheme for the agent with DID handling, verifiable credentials (VC), and verifiable presentations (VP). Each of the agents collaborates on a shared workspace based on blackboard to handle tasks to reach a goal. Finally, we demonstrate its feasibility through a proof-of-concept (PoC) for

Journal of Web Engineering, Vol. 25_1, 1–18.
doi: 10.13052/jwe1540-9589.2511

Agentic AI service architecture. This proof-of-concept, structured across Phase 1 (discovery, verification, and scoped authorization) and Phase 2 (problem posting and blackboard-mediated collaboration), demonstrates that DID-backed credentialing can securely support multi-agent execution under a least-privilege operational model.

Keywords: Agentic AI, service-oriented architecture, DID, verifiable credentials, verifiable presentations.

1 Introduction

Service-Oriented Architecture (SOA) [1], is a software design approach that structures an application as a collection of modular, independent, and reusable components called services. Instead of building a single, monolithic application, developers create distinct services that perform specific business functions, and these services communicate with each other over a network to achieve a larger goal.

In a SOA approach, you would break down the complex tasks of building an autonomous agent. Those tasks build into distinct and independent services. Each service performs a specific function and is designed to be reusable by other components. An autonomous agent is a high-level component in the perspective of SOA that takes a complex task and breaks it down into smaller, manageable steps.

The agent acts as a dynamic orchestrator that, upon receiving a task, applies agentic reasoning and available capabilities, including other agents, to determine the most effective sequence of actions. Rather than executing a predefined script, it decomposes the task into subtasks and uses a set of services and tools to achieve the goal.

To represent an agent with DID as a component in SOA, we define the agent as a self-contained, reusable service with a well-defined API. Digital Identity (DID) is a decentralized identity system that allows individuals and organizations to create and control agents with their own digital identifiers.

This approach based on DID is used to handle the agent not as an internal function, but as an external service that other applications can call to perform a specific task.

To trace the capabilities of an agent deployed in a SOA, you need to monitor the requests and responses at both the service level and the internal agent level. SOA allows us to understand what the agent can do, how it's performing, and what steps it's taking to fulfil a request.

In this paper, we present the design of a verifiable-credentials (VC) architecture and demonstrate its feasibility via a proof-of-concept (PoC) that implements end-to-end credential issuance and verification using DIDs, issuer-signed JWT credentials, and selective disclosure. Building on this foundation, we design an agentic AI task-delegation process that solves complex problems by decomposing them into subtasks executed by multiple agents coordinated through a global blackboard.

The process follows a two-phase discipline: Phase 1 (Discovery, Verification, Scoped Authorization) verifies agent capability with a VP challenge and grants short-lived, scope-limited access; Phase 2 (Problem Posting and Blackboard-Mediated Collaboration) posts the task to the shared workspace, where verified agents contribute hypotheses/evidence and converge under policy-based selection. This combination enforces least-privilege access, preserves auditability and revocation awareness, and enables scalable, composable multi-agent cooperation within an SOA-aligned framework.

2 Related Work

2.1 Service-Oriented Architecture (SOA)

When applying SOA, the functionalities of existing applications are bundled into functional units with business meaning and implemented as services through a standardized calling interface. SOA refers to a software development architecture that composes applications with services according to a company's business tasks [2].

The core idea of SOA is to break down a large application into smaller, manageable, and interconnected services. This approach is guided by several key principles [1–4]:

- Loose Coupling: Services are designed to be independent, with minimal knowledge of each other's internal workings. A service consumer only needs to know the service's interface and the data it expects and returns. This means you can update or replace a service without affecting other parts of the system.
- Reusability: Services are designed to be generic and reusable across different applications. For instance, a single "customer authentication" service can be used by a company's website, mobile app, and internal tools.
- Interoperability: Services communicate using standardized protocols (like HTTP/REST or SOAP), which allows them to be developed in

different programming languages and run on different platforms. This enables seamless integration between various systems.
- Discoverability: Services are registered in a central directory or service registry, which acts as a "phone book" for the architecture. This allows other applications to easily find and use available services without needing to know their specific location.

SOA workflow involves three main roles [2]:

- Service Provider: This is the entity that creates and hosts a service.
- Service Registry: A central catalogue where service providers publish their services and their capabilities.
- Service Consumer: An application or another service that needs a specific function. It queries the service registry to find the right service and then communicates with the service provider to use it

2.2 Decentralized Identifiers (DID)

The World Wide Web Consortium's Decentralized Identifiers (DIDs) specification defines a decentralized, standards-based model for digital identity [5, 6]. A DID is a globally unique identifier that resolves – via a verifiable data registry such as a blockchain or other distributed system – to a DID Document containing public keys, verification methods, and service endpoints, thereby removing any single, central authority from the trust path. This DID specification aligns naturally with autonomous and agent-based architectures: intelligent service agents in SOA-based or agentic AI systems can use DIDs to authenticate peers, establish trusted communication channels, and transact securely across heterogeneous environments. In combination with verifiable credentials, DIDs enable interoperability, verifiability, and accountability at scale.

2.3 Blackboard Architecture

The blackboard architecture enables heterogeneous knowledge sources to post partial solutions and evidence to a shared workspace, while a controller opportunistically selects the next action. Foundational accounts formalize the tripartite structure – blackboard, knowledge sources, and control – and explain how bottom-up and top-down reasoning interact to converge on solutions under uncertainty [7–10]. Empirical roots such as Hearsay-II demonstrated how multi-level hypotheses (from phonemes to semantics) can compete and integrate on a common board, establishing the paradigm's

practical value [11]. Subsequent work compared control strategies – priority-based, utility-driven, and parallel scheduling – highlighting the importance of strategy selection for domain-specific performance [8].

Recent research has adapted the blackboard paradigm to LLM-based multi-agent systems (MAS), using a shared memory/board to coordinate role-specialized agents that iteratively propose hypotheses, critique partial results, and refine plans. In particular, Li et al. (2025) [12] explore advanced MAS designs grounded in a blackboard architecture, reporting gains in end-to-end task success by leveraging round-based contribution and selective disclosure of intermediate artifacts. This line of work aligns closely with our SOA-based intelligent service-agent framework: services map to knowledge sources, the registry mediates verification and dispatch, and the blackboard provides a compositional substrate for cooperative problem solving with verifiable, least-privilege contributions [7, 12].

3 Agent Based Service Architecture

In the context of intelligent service agents and agentic AI, DIDComm [13, 14] equips autonomous agents with an interoperable, end-to-end encrypted messaging layer. Building on this, an SOA view of LangChain [15] and MCP [16] encapsulates each tool or chain as a service with a well-defined interface, enabling agents to compose capabilities dynamically while preserving loose coupling and reuse. This approach enhances modularity, reusability, and scalability.

- Reusability: The same agent service can be used by multiple different applications. For example, a "customer support agent" service could be called by a mobile app, a web chat widget, and an internal tool for a human support agent.
- Scalability: The agent service can be scaled independently of the applications that consume it. If the agent is under heavy load, you can deploy more instances of just the agent service without affecting other parts of your system.
- Maintainability: You can update or replace the agent's internal implementation (e.g., switching from one LLM to another) without requiring any changes to the applications that use it. If the service's API contract remains the same, the consumers of the service will not be affected.
- Interoperability: Agents can be implemented in any technology or programming language and exposed via standard HTTP/JSON (or

gRPC) so that any client capable of making HTTP requests can interact with them. For agent-to-agent workflows, we use DIDComm v2 as a transport-agnostic, end-to-end-encrypted protocol that works across vendors and DID methods. VC/VP proofs, capability tokens, and job/control messages are exchanged as DIDComm messages, with mediation/routing and offline delivery support, enabling heterogeneous agents to interoperate without a shared runtime or trusted network. Access control remains consistent via verifiable presentations and short-lived, scope-limited authorization tokens conveyed over DIDComm.

3.1 Agentic AI registration Process

Figure 1 shows the provisioning phase in which an agent is instantiated, assigned a decentralized identifier (DID), and authorized through an authority token. Three roles participate: the Agent Creator (issuer/CA-like authority), the Agent (autonomous executor), and the Agent Registry (verifier and dispatcher).

1. Create (Agent initialization). The Agent Creator initializes a new agent instance. At this point the agent is an uninitialized entity with no identity or privileges.
2. Create key pair (issuer key). Acting in a certificate-authority–like role, the Agent Creator locally generates and retains exclusive control of an

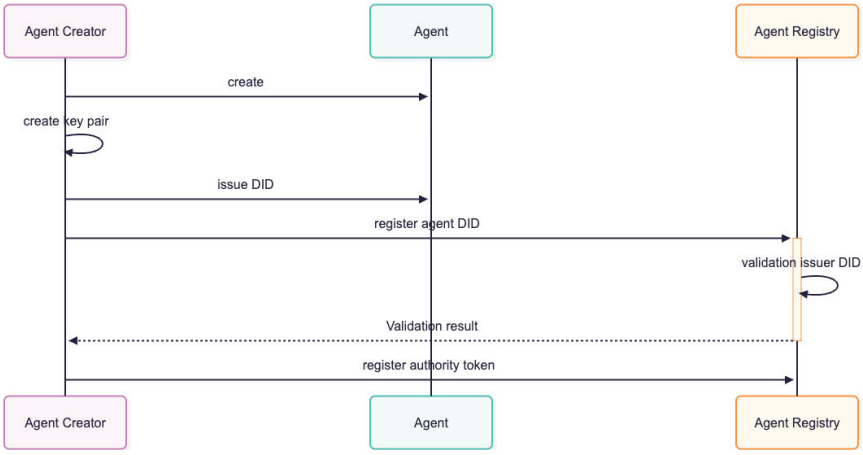


Figure 1 Agentic AI registration process.

issuer key pair. The private key will later be used for issuing and signing DID-related artifacts and authority tokens.
3. Issue DID. The Agent Creator issues a DID for the agent and constructs the corresponding DID document under the Creator's control. This binds the agent's identity to the Creator's issuer key and governance policies.
4. Register agent DID. The Agent (or the Creator on its behalf) registers the agent DID with the Agent Registry to enable discovery and later authorization queries.
5. Validation (issuer/DID checks). The Agent Registry resolves the DID and validates the issuer/authority, including signature verification and policy checks over the Creator's identifier.
6. Validation result. The Registry returns the validation outcome to the Agent Creator, establishing a trusted state for subsequent authorization.
7. Register authority token. The Agent Creator registers an authority token at the Registry. The token encodes the agent DID and its capabilities/scopes (e.g., data-access class, resource constraints, expiry), thereby authorizing the agent to request and execute jobs under least-privilege constraints.

After step 7, the Registry holds a validated mapping between the agent DID and a signed authority token issued by the Agent Creator. This completes identity bootstrapping and sets the preconditions for later phases (proof request, VP presentation, and job dispatch).

3.2 Agentic AI Task Delegation Process

We structure the agentic AI task-delegation process into two phases. Phase 1 (Discovery, Verification, and Scoped Authorization) identifies eligible agents and grants least-privilege access: the Service Requester issues a capability-constrained dispatch request; the Agent Registry challenges candidate agents for a verifiable presentation (VP); any missing capability credentials are issued; agents assemble and present VPs; and, upon successful verification (signatures, DID binding, status/revocation), the Registry issues short-lived, scope-limited blackboard access tokens. Phase 2 (Problem Posting and Blackboard-Mediated Collaboration) coordinates cooperative problem solving: the Service Requester posts the problem to the blackboard; verified agents subscribe, read, and contribute hypotheses/evidence under their scoped permissions; the Registry (or an arbiter) evaluates claims by policy (confidence, cost/time, reputation) until convergence; and the final result is

returned with an auditable trail. This two-phase structure preserves a prove–reveal–dispatch discipline while enabling scalable, verifiable multi-agent collaboration.

3.2.1 Phase 1 – Discovery, verification, and scoped authorization

Figure 2 presents details about Phase 1 – Discovery, Verification, and Scoped Authorization of the agentic AI task-delegation process, showing how the registry identifies eligible agents, verifies their capabilities via verifiable presentations (VPs), and grants short-lived, scope-limited access to the blackboard for subsequent collaboration.

1. Capability-constrained dispatch request. The Service Requester asks the Agent Registry to dispatch a task with a specified capability requirement and relevant constraints (e.g., deadline, data policy).
2. VP challenge to candidates. The Registry selects eligible candidate agents and issues a verifiable presentations(VPs) challenge to each (nonce- and audience-bound).
3. Capability credential issuance (if needed). Agents lacking current proofs request a capability credential from the Agent Creator; the Creator issues the credential (e.g., VC or SD-JWT).

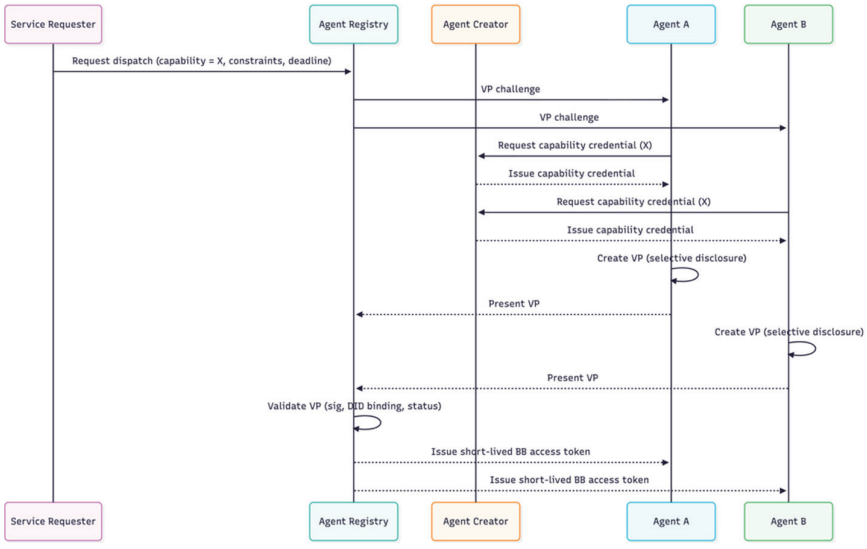


Figure 2 Discovery, verification, and scoped authorization.

4. Presentation assembly and submission. Each Agent prepares a VP with selective disclosure and presents it to the Registry.
5. Verification and authorization. The Registry verifies signatures, DID binding, and status/revocation. Upon success, it issues a short-lived, scope-limited blackboard access token to each verified Agent.

Outcome. Only VP-verified agents receive time-bounded, scope-restricted authorization to access the blackboard; no agent list is exposed to the Service Requester.

3.2.2 Phase 2 – Problem posting and blackboard-mediated collaboration

Figure 3 shows details Phase 2 – Problem Posting and Blackboard-Mediated Collaboration of the agentic AI task-delegation process, showing how a shared workspace coordinates cooperative problem solving among verified agents under least-privilege controls.

1. (Problem posting) The Service Requester posts the problem to the Blackboard Service (inputs, constraints, trace ID).
2. (Subscription and intake) Verified Agents subscribe to the blackboard feed and read the problem; each access is checked against the short-lived, scope-limited token from Phase 1.
3. (Contributions) Agents contribute hypotheses and evidence/artifacts under their Agent DID, signing each contribution and adhering to scope constraints (e.g., bb:read, bb:write:hypothesis, bb:write:claim).
4. (Policy-based selection and convergence) The Agent Registry (or an arbiter) evaluates claims by policy – confidence, cost/time, reputation – and iterates if needed until a best claim is selected.

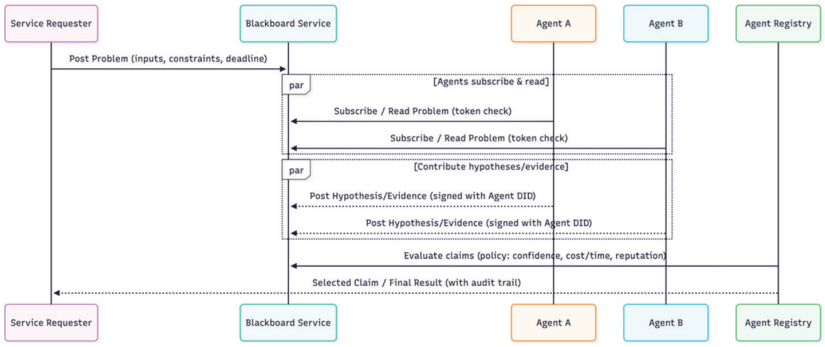


Figure 3 Problem posting and blackboard-mediated collaboration.

5. (Returning the result) The Registry returns the final result to the Service Requester with an auditable trail (trace ID, proofs, and contribution lineage).

As illustrated in Figures 2 and 3, the workflow adheres to a prove–reveal–dispatch discipline: credentials are verified before any access is granted, authorizations are least-privilege and time-bound, and collaboration occurs via verifiable, loosely coupled contributions.

4 Experiment

4.1 Experimental Setup

Our experimental environment comprises four components:

- Registry (Verifier/Dispatcher). Implemented with ACA-Py [17]; verifies VPs, checks status/revocation, and issues short-lived, scope-limited authorization tokens for blackboard access.
- Agent (Holder/Executor). Assembles and presents VPs that satisfy capability requirements, then executes assigned tasks (single-agent or collaborative).
- Creator (Issuer). Issues capability credentials (JWT-VC or JSON-LD VC) bound to the agent's DID; publishes the issuer key via did:web for verification.
- Blackboard Service. HTTP-exposed, append-only workspace where problems, hypotheses, and claims are posted; supports policy-based selection and auditing.

All agents run in a containerized environment with ACA-Py (v1.2 LTS) and no ledger. Agents use did:key, the issuer is published as did:web. Experiments run on a local Docker bridge network.

Phase 1 establishes discovery, verification, and scoped authorization. The Registry challenges candidate agents with a VP request that includes a nonce (challenge) and an audience (domain). Agents present proofs of capability using ACA-Py Present-Proof v2 (DIF/PE) based on credentials issued by the Creator (JWT-VC or JSON-LD VC). The Registry verifies issuer signatures, DID binding, and credential status or revocation, then grants least-privilege access by issuing a short-lived, scope-limited authorization token for blackboard operations. This phase ensures that only verified agents obtain time-bounded permissions before any problem is posted or collaboration begins.

Phase 2 evaluates multi-agent coordination on an image-classification micro-task (100–1,000 images), decomposed into a pipeline of preprocess, coarse labeling, specialist checking, and claim submission. This structure lets us measure end-to-end performance while observing how the blackboard mediates hypothesis sharing and policy-based convergence.

4.2 Phase 1: Discovery, Verification, and Scoped Authorization

The following is the scenario for prove the designed operational scheme:

1. The Agent Creator launches a new agent (ACA-Py) and initializes it as an unprivileged entity.
2. The Creator issues a DID for the agent and publishes the corresponding DID document as required by the chosen method.
3. Figure 5 illustrates an issued token with the description of capability. The Creator generates and signs the agent capability token as a JWT

```
(base) dbc@dbc-gpu:~/project/agent$ curl -s -X POST http://localhost:8021
/wallet/did/create -H "Content-type: application/json" -d '{"method":"key
"}'
{"result": {"did": "did:key:z6MkwUAYf4Bd8Az37mMRTTW4AhzBrk7EXS4TcfxT5yaSF
JLh", "verkey": "J1uW4owBndVa1GWimtYDKcSC3AqP7Yp6vf3XFhcRL5ZK", "posture"
: "wallet_only", "key_type": "ed25519", "method": "key", "metadata": {}}}
```

Figure 4 DID issuance.

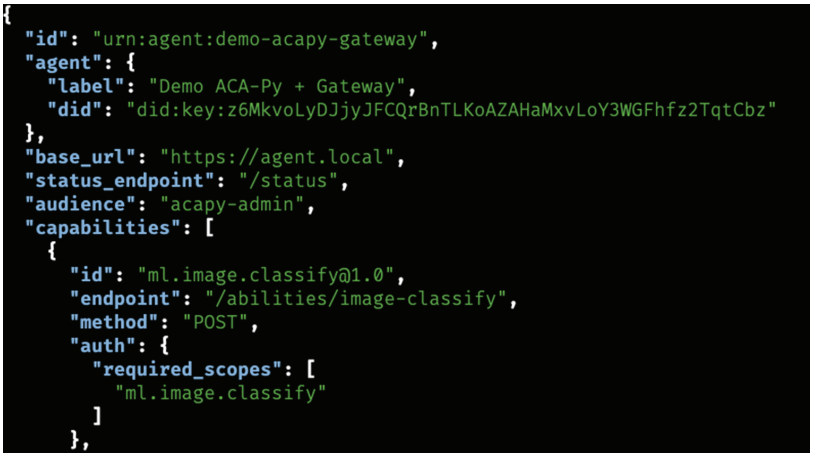
```
# issue_vc.py
import time, jwt
from pathlib import Path

priv = Path("creator_es256_priv.pem").read_text()
now = int(time.time())

payload = {
  "iss": "did:web:creator.example",
  "sub": "did:key:zAgent123",
  "aud": "https://registry.example.org",
  "nbf": now,
  "exp": now + 24*60*60,
  "vc": {
    "type": ["VerifiableCredential","AgentCapabilityCredential"],
    "credentialSubject": {
      "id": "did:key:zAgent123",
      "capabilities": ["image.classify"]
    }
  }
}
headers = {"alg":"ES256","kid":"did:web:creator.example#key-1","typ":"JWT"}

token = jwt.encode(payload, priv, algorithm="ES256", headers=headers)
print(token)
```

Figure 5 Capability token issuance.



```
{
  "id": "urn:agent:demo-acapy-gateway",
  "agent": {
    "label": "Demo ACA-Py + Gateway",
    "did": "did:key:z6MkvoLyDJjyJFCQrBnTLKoAZAHaMxvLoY3WGFhfz2TqtCbz"
  },
  "base_url": "https://agent.local",
  "status_endpoint": "/status",
  "audience": "acapy-admin",
  "capabilities": [
    {
      "id": "ml.image.classify@1.0",
      "endpoint": "/abilities/image-classify",
      "method": "POST",
      "auth": {
        "required_scopes": [
          "ml.image.classify"
        ]
      },
```


Figure 6 Agent registry catalogue.

containing, at minimum: iss = Creator DID, sub = Agent DID, capabilities (e.g., ["ml.image.classify"]), aud = Agent, iat/exp for bounded lifetime, and a nonce/identifier for replay protection.

4. In registration phase, the Creator registers the JWT with the Agent Registry as shown in Figure 6. The registry validates the signature and claims (issuer DID, audience, expiry, scope) and records the mapping from Agent DID to authorized capabilities.

4.3 Phase 2: Problem Posting and Blackboard-Mediated Collaboration

The following scenario demonstrates the execution of Phase 2, showing how verified agents interact through the Blackboard Service using the scoped authorization token issued in Phase 1.

1. The Service Requester submits the task to the Blackboard Service (Figure 8), including the input payload, operational constraints, and the trace identifier. The blackboard stores the task as an append-only entry accessible only to agents holding valid short-lived tokens from Phase 1.
2. Agents A and B, whose VPs are successfully validated in Phase 1, subscribe to the blackboard event stream. Each read request is evaluated against the scope-limited authorization token (e.g., bb:read). Unauthorized or expired access attempts are rejected by the Blackboard Service (Figure 9).

```
% cat verity.py
import jwt, json, requests, sys

REGISTRY = "http://localhost:9100"
cat_url = f"{REGISTRY}/capabilities/agent-1.json"
sig_url = f"{REGISTRY}/capabilities/agent-1.jws"
jwks_url = f"{REGISTRY}/.well-known/jwks.json"

cat = requests.get(cat_url).json()
jws = requests.get(sig_url).text.strip()
jwks = requests.get(jwks_url).json()

key = jwt.PyJWKClient(jwks_url).get_signing_key_from_jwt(jws).key
claims = jwt.decode(jws, key, algorithms=["RS256"])

assert claims == cat, "Catalog and JWS payload differ!"
print("OK: signature valid and payload matches catalog.")

(jwt) dbc@DBC-mac ~/acapy-test/verify_catalog
% python verity.py
OK: signature valid and payload matches catalog.
```

Figure 7 Verification.

```
% curl -X POST http://localhost:8100/tasks \
  -H "Content-Type: application/json" \
  -d '{
        "trace_id": "phase2-exp-001",
        "dataset_id": "imagenet-mini-0001",
        "images": [
          {
            "image_id": "img-000001",
            "uri": "https://example.com/images/000001.jpg",
            "metadata": {"split": "test", "source": "s3://bucket/path/000001.jpg"}
          },
          {
            "image_id": "img-000002",
            "uri": "https://example.com/images/000002.jpg",
            "metadata": {"split": "test"}
          }
        ],
        "constraints": {
          "max_images": 1000,
          "global_timeout_sec": 600,
          "max_parallel_agents": 8
        }
      }'

{"trace_id":"phase2-exp-001","dataset_id":"imagenet-mini-0001","images":[{"image_id":"img-000001","uri":"
https://example.com/images/000001.jpg","metadata":{"split":"test","source":"s3://bucket/path/000001.jpg"}
},{"image_id":"img-000002","uri":"https://example.com/images/000002.jpg","metadata":{"split":"test"}}],"c
onstraints":{"max_images":1000,"global_timeout_sec":600,"max_parallel_agents":8},"stages":[{"name":"prepr
ocess","description":"Image preprocessing","policy":null},{"name":"coarse_label","description":"Coarse La
beling","policy":null},{"name":"specialist_check","description":"Specialist Check","policy":null},{"name"
:"claim_submission","description":"Claim Submission","policy":null}],"id":"1ca286cf-4288-4f8a-ae59-cdd02e
fb4937","created_at":"2025-11-26T10:16:42.935116","status":"pending","current_stage":"preprocess"}%
```

Figure 8 Submit the task to blackboard service.

3. Upon reading the task, each agent processes the input according to its capabilities – for example, image preprocessing or coarse classification – and submits results to the blackboard. Each submission includes the agent's DID, a cryptographic signature, the applicable capability

Figure 9 Get task by authorized token.

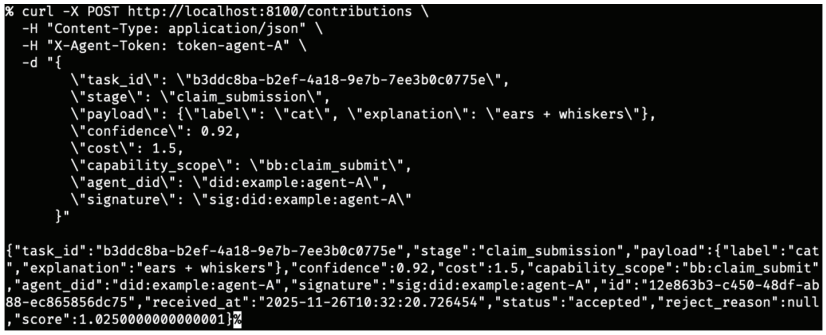


Figure 10 Signed agent contribution.

scope, and a timestamp. These signed contributions form an auditable chain of intermediate steps (Figure 10).

4. The blackboard controller evaluates contributions based on confidence, latency, cost, and agent reputation. The arbiter iterates through multiple evaluation rounds as new hypotheses arrive, discarding out-of-scope or low-confidence entries. The process continues until the final claim meets convergence requirements.
5. The selected claim is packaged with the trace ID, verification metadata, capability proofs, and the full lineage of hypotheses. The Service Requester receives not only the final output but also a verifiable audit trail showing how the collaborating agents produced the result under scoped authorization.

5 Result and Discussion

The proof-of-concept successfully demonstrated agent registration, retrieval, and verification. However, for large-scale operations involving interactions

among multiple agents, additional security considerations are required. Such inter-agent collaboration can be facilitated through DIDComm [1, 2]. The proof-of-concept implementation successfully demonstrated the end-to-end workflow of the proposed scheme, including agent instantiation, DID issuance, capability-token generation, registry validation, and verifiable-presentation–based authorization. The results confirm that the designed SOA-aligned operational model can enforce least-privilege access while maintaining auditability across the full agent lifecycle. In particular, the experiment shows that short-lived, scope-restricted tokens – issued only after VP verification – are sufficient for securing multi-agent task execution within a blackboard-mediated environment.

However, several considerations emerge when scaling the system to larger deployments involving many heterogeneous agents. First, coordination overhead increases as the number of contributions grows, requiring more advanced policy-based arbitration or distributed control strategies. Second, inter-agent communication must remain secure and interoperable across different execution environments and DID methods. These challenges can be addressed by incorporating DIDComm-based secure channels, which provide authenticated, encrypted, and routable messaging suitable for cross-domain agent interactions [1, 2].

Overall, the PoC validates the feasibility of the proposed credential-driven agentic AI architecture while highlighting areas – such as trust negotiation, routing, and scalable blackboard control – that warrant deeper investigation for real-world, multi-agent ecosystems.

6 Conclusion

In this paper, we presented an SOA-based operational scheme for agentic AI systems, leveraging decentralized identifiers (DIDs) to enable verifiable credentials throughout the lifecycle of agent interaction and task execution. The proposed architecture consists of two core processes: the Agentic AI registration process, which establishes identity and capability through DID-based credentials, and the task-delegation process, which coordinates multi-agent collaboration via scoped authorization and a blackboard-mediated workflow. A proof-of-concept prototype demonstrates the feasibility of the approach and validates that DID-backed credentialing can securely support multi-agent execution under least-privilege constraints.

For future work, we aim to extend this framework toward building trustworthy service agents capable of negotiating products and services

within a multi-agent environment. This includes handling transactional workflows, dynamic capability exchange, and broader interoperability, ultimately enabling secure and autonomous service ecosystems.

Acknowledgement

The present research was supported by the research fund of Dankook University in 2025.

References

[1] Erl, T. (2005). Service-oriented architecture: concepts, technology, and design. Prentice Hall PTR.

[2] Papazoglou, M. P., and Van Den Heuvel, W. J. (2007). Service oriented architectures: approaches, technologies and research issues. The VLDB journal, 16(3), 389–415.

[3] Josuttis, N. M. (2007). SOA in practice: the art of distributed system design. O'Reilly Media, Inc.

[4] MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., and Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. OASIS standard, 12(S18), 1–31.

[5] Reed, D., Sporny, M., Longley, D., Allen, C., Grant, R., Sabadello, M., and Holt, J. (2020). Decentralized identifiers (dids) v1. 0. Draft Community Group Report.

[6] Yao, W., Du, W., Gu, J., Ye, J., Deek, F. P., and Wang, G. (2024, July). Establishing a baseline for evaluating blockchain-based self-sovereign identity systems: A systematic approach to assess capability, compatibility and interoperability. In Proceedings of the 2024 6th Blockchain and Internet of Things Conference (pp. 108–119).

[7] Nii, H. P. (1986). The blackboard model of problem solving and the evolution of blackboard architectures. AI Magazine, 7(2), 38–53.

[8] Carver, N., and Lesser, V. R. (1994). The evolution of blackboard control architectures. Expert Systems with Applications, 7(1), 1–30.

[9] Craig, I. D. (1988). Blackboard systems. Artificial Intelligence Review, 2(2), 61–93.

[10] Engelmore, R., and Morgan, T. (Eds.). (1988). Blackboard systems. Addison-Wesley.

[11] Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. (1980). The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. ACM Computing Surveys, 12(2), 213–253.
[12] Li, X., et al. (2025). Exploring advanced LLM multi-agent systems based on blackboard architecture (arXiv:2507.01701).
[13] Curren, S., Looker, T., and Terbu, O. (2022). DIDComm messaging v2. x editor's draft. DIF. Available online: https://identity.foundation/didcomm-messaging/spec/ (accessed on 16 January 2024).
[14] De Prisco, R., Shevchenko, S., and Faruolo, P. (2024). Enhancing OpenID connect for verifiable credentials with DIDComm. In Proceedings of the 21st international conference on security and cryptography (SECRYPT 2024) (pp. 844–849).
[15] Mavroudis, V. (2024). LangChain.
[16] Hou, X., Zhao, Y., Wang, S., and Wang, H. (2025). Model context protocol (mcp): Landscape, security threats, and future research directions. arXiv preprint arXiv:2503.23278.
[17] OpenWallet Foundation. (2024). _Aries Cloud Agent – Python (ACA-Py)_. Available at: https://github.com/openwallet-foundation/acapy.

Biographies

Dong Bin Choi received the bachelor's degree in computer engineering from KNUE in 2011, the master's degree in computer science from Dankook university in 2018, and in the course of philosophy of doctorate degree in Computer Science Dankook University, currently working on AI and blockchain.

Yunhee Kang earned a BS in Computer Engineering (1989) and an MS in Computer Engineering (1993), both from Dongguk University in Seoul, Korea. He received a PhD in Computer Science (2002) from Korea University in Seoul, Korea. He has been working as a Full Professor at Baekseok University in Cheonan, Korea since March 2002. His research interests include Trusted Computing, Cloud computing, Applied AI, Blockchain and Web3.

Young B. Park received the bachelor's degree in computer engineering from Sogang University in 1985, the master's degree in computer science from NY Polytechnic University in 1987, and the philosophy of doctorate degree in Computer Science NY Polytechnic University in 1991, respectively. He is currently working as a Professor at the Department of Software Dankook University. His research areas include blockchain, deep learning, and software engineering. He has been serving as a reviewer for many highly respected journals.