Choreographies, DSUs, SVDs for Multi-Agent Systems

Author: Sînică Alboaie, PhD, Axiologic Research
Purpose: A report explaining in a simple language Concepts: Choreographies, DSUs, SVDs,etc
Visibility: Public
Date: 2023
Version: 0.1

Executive Summary

This report presents the most relevant conceptual results and insights obtained over the last 15 years by Axiologic Research in collaboration with various researchers and as part of many research initiatives. We will start presenting our research on executable choreographies based on swarm communication derived from projects like SwarmESB and PrivateSky. We also delve into explaining the KeySSI (Key Self Sovereign Identities), DSU (Data Sharing Unit) and SVD (Self Validating Data) concepts as simply as possible. At the end of the report, we will talk about the potential to connect SVDs with executable choreographies. The SVD and executable choreographies merging were never implemented and could be a substantial project for us in the following years.

Our goal for this report is to provide examples that are as simple and clear as possible without delving into specific technologies or implementations, instead focusing on illustrating the concepts. The examples will be related to AI systems because the potential for these innovations to be implemented on a large scale is even more significant beyond the area where we have the most experience in enterprise blockchain systems for Digital Trust Ecosystems. There's a substantial need for these ideas for creating verifiable systems with security by design for complex multi-agent systems.

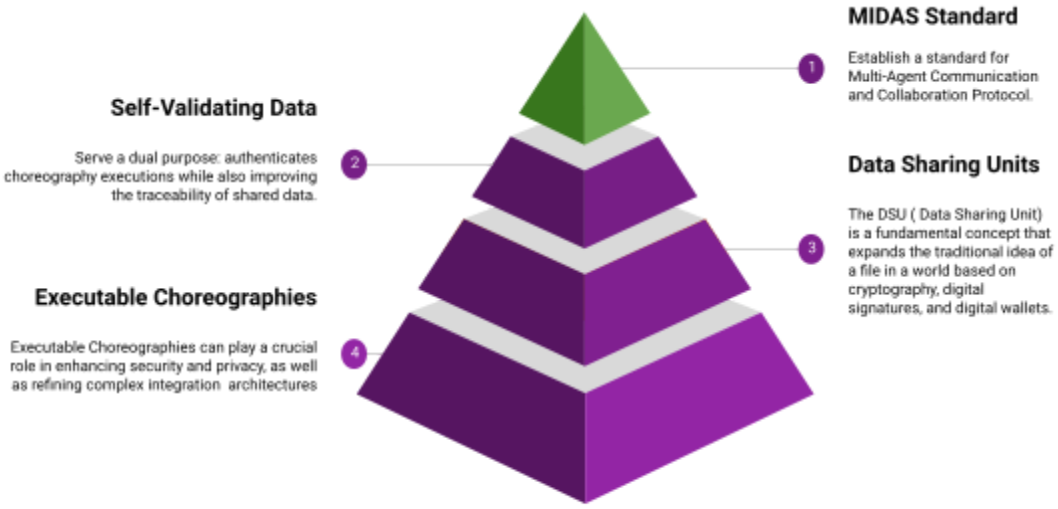

Diagram 1: Major Results of Our Research

Our primary research directions and inventions over the last 15 years are progressively converging into MIDAS, which aims to become a new technological standard for multi-agent systems based on AI agents. MIDAS aims to provide a layer for real-time communication and document processing for AI-based agents. Executable Choreographies could significantly enhance security and privacy and refine complex integration architectures as multi-agent architectures.

Subsequently, we present the OpenDSU concepts, which introduce fundamental concepts that broaden the traditional idea of a file in a cryptography-empowered world, incorporating digital signatures and digital sovereignty through wallets. More recently, we have started to appreciate the potential of Self-Validating Data (SVD), which serves a dual purpose: it authenticates choreography executions while enhancing the traceability of shared data.

Executable Choreographies Introduction

Diagram 2: A Choreography is a custom communication protocol between different nodes

The concept of "choreography" is about having a formalised description of a communication protocol between independent entities. For instance, detailing the inputs and outputs for a web API constitutes the simplest form of choreography.

Diagram 3: Generalising the API call concept creates the concept of "choreography execution"

An API call facilitates data flow between a client and a server and then back to the client. This is like a "choreography execution".

The "choreography" concept becomes more valuable when multiple entities are engaged in a singular communication "flow", allowing for discussions on choreographies or orchestrations.

Orchestration refers to scenarios where a central conductor manages all subsequent calls and aggregates (composes) their results in a unified result for the original requester.

In contrast, "choreography" applies when no central conductor exists, and the entities coordinate their interactions independently.

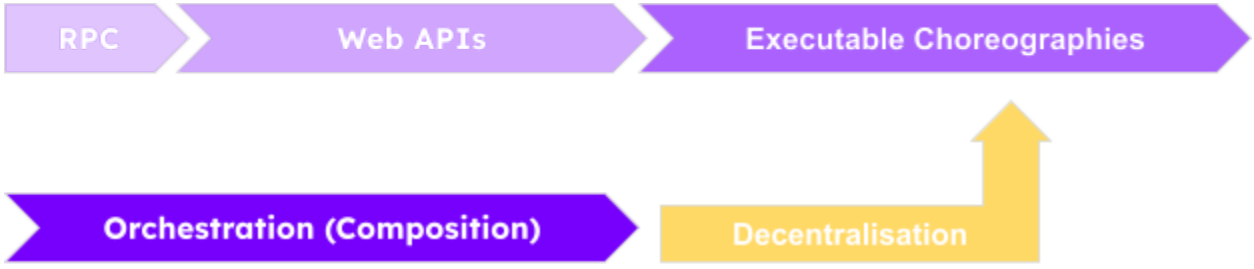

Diagram 4: The evolution of RPC concept toward Executable Choreographies

An "executable choreography" represents a choreography description (a flow or communication protocol description) represented in the form of an actual software program (a dynamic script) that can be executed across multiple network nodes. This approach enables complex interactions across a distributed network, emphasising the autonomous and coordinated execution of processes without needing a centralised controller.

Implement Executable Choreographies using “Swarm Communication” (“callflows”)

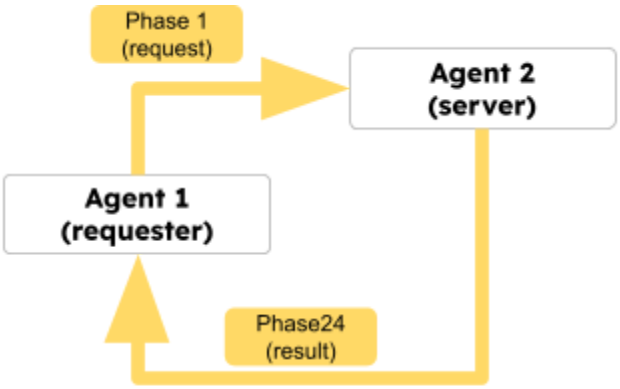

Diagram 5: The simples “flow” of communication between 2 agents (Agent 1, Agent 2)

The concept of "swarm communication" [2], also sometimes named in our work as “callflows”, is just a method for implementing "executable choreographies" [3]. In swarm communication, each phase of the communication flow is represented as code associated with locations of execution. The swarm communication comes with the intuition that the model of the computational flows is a jump between independent nodes, each gathering new insights about each node. It is like an RPC call from a client, but instead of hitting only one server, there is a series of executions in multiple nodes, and eventually, the results are returned to the client. Even more complex “communication patterns” could be described; the linear one is the simplest.

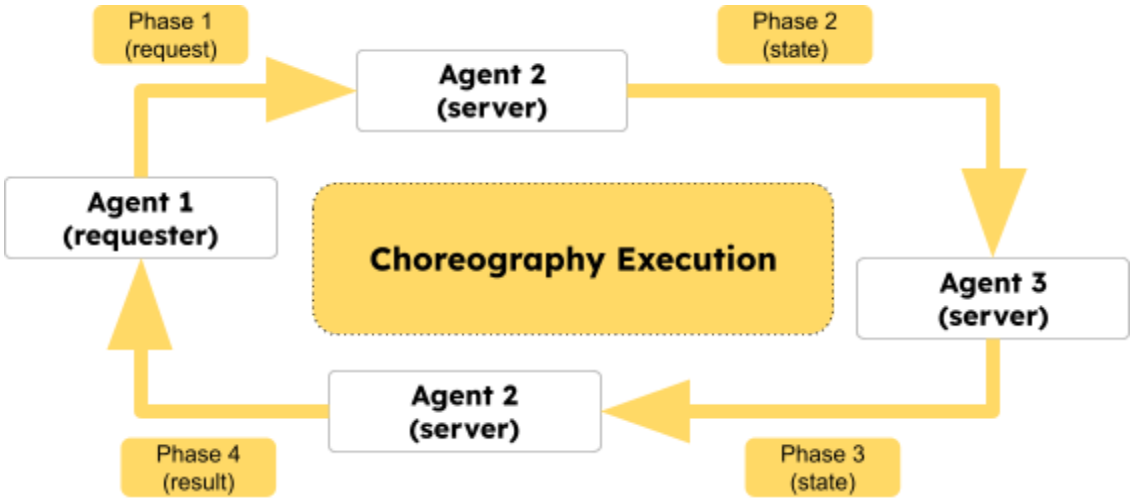

Diagram 6: A simple “flow” of communication between 3 agents

Each jump in execution will also carry some data obtained in each node; therefore, the execution of choreography has some state that is transferred with each execution phase.

Swarm Communication (Implementation Independent Examples)

The following diagram presents code in an "imagined" choreography-ready programming language. In our implementation, we used internal DSL based on JSON structure. Still, for clarity, we present code of some "hello world" examples to illustrate the programming with "swarm communication" based executable choreographies. Other types of executable choreographies exist and will have slightly different properties if they are not based on "swarm communication"

The code outlines an executable choreography named HelloWorld2, commencing with a request phase at agent1 where "this.data" is initialised as "null" and a transition to agent2 is prepared. In the subsequent phase, inAgent2, executed at agent2, "this.data" is assigned the string "Hello From Agent2". Finally, the choreography concludes with a response phase back at agent1, where the updated "this.data" value is logged to the console, demonstrating a simple yet effective mechanism for data exchange and processing across distributed agents.

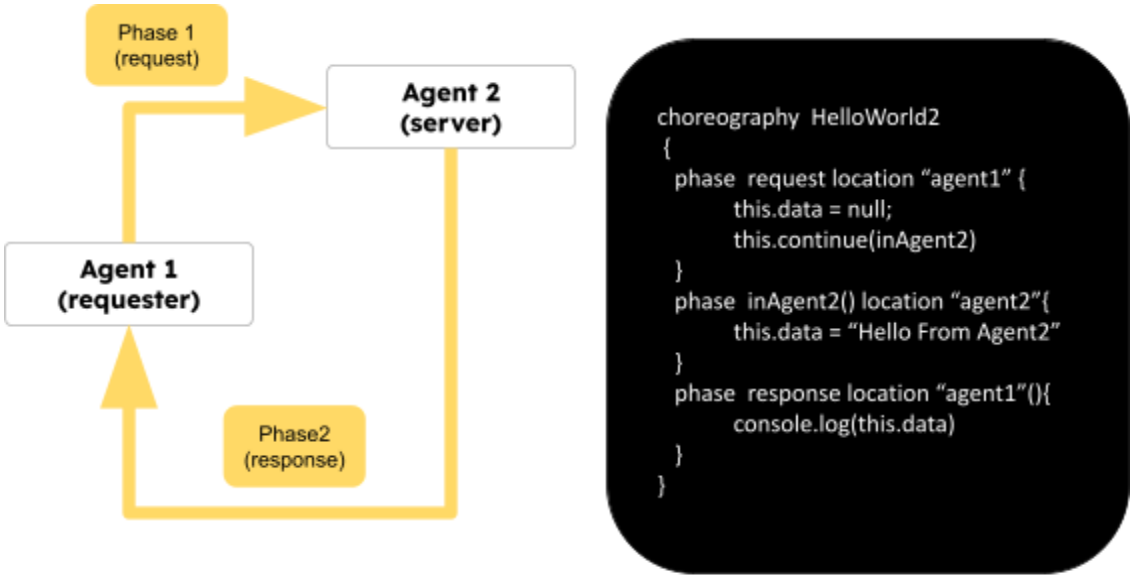

Diagram 7: The simplest "swarm communication" choreography between 2 agents

In HelloWorld2 choreography, the process begins with Agent1, which initialises the operation without any preliminary data and delegates the task to Agent2. Agent2 then takes over, crafting the message "Hello From Agent2" in a single step. After completing its part, Agent2 passes control back to Agent1, which concludes the operation by outputting the message. This choreography embodies a straightforward interaction between two agents, focusing on a direct handover of tasks with minimal complexity.

The following diagram introduces another "hello world" example for choreographies. Both HelloWorld2 and HelloWorld3 choreographies represent how tasks are distributed and executed across multiple agents within a distributed computing environment, using our imaginary programming language to highlight the concept of executable choreographies. However, transitioning to HelloWorld3, the scenario becomes more intricate. The choreography still commences with Agent1 but quickly evolves as Agent2 adds only "Hello" to the message, showcasing an initial collaboration layer. The narrative progresses with the involvement of Agent 3, which appends " World" to the evolving message, indicating an extended chain

of collaboration across different agents. Subsequently, the baton is passed back to Agent2, which adds a final flourish with "!" before being handed over to Agent1 to complete the process.

This sequence illustrates a more complex and nuanced approach to constructing distributed computation, involving multiple agents and several steps that highlight the choreography's capacity for detailed, distributed message assembly.

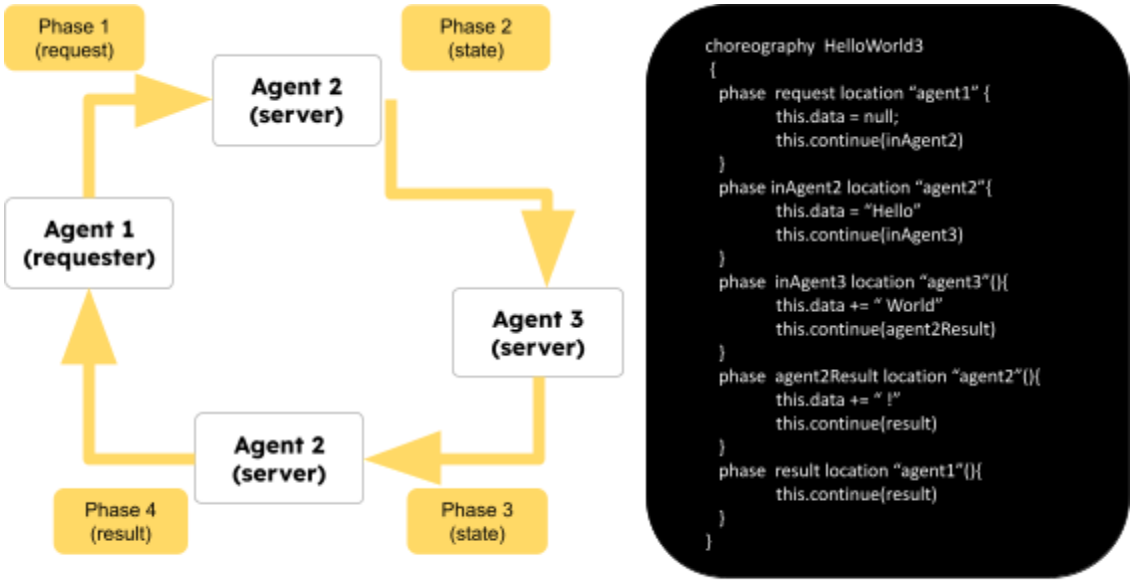

Diagram 8: A "flow" between 3 agents (Agent 1, Agent 2, Agent 3)

The primary distinction between HelloWorld2 and HelloWorld3 lies in their approach to obtaining the result. HelloWorld2 presents a more straightforward, linear interaction between two agents, emphasising straightforward task delegation and completion. Conversely, HelloWorld3 unfolds a more complex narrative involving multiple agents in a collaborative effort to construct a message, illustrating the potential of executable choreographies in orchestrating complex interactions across a distributed system. This complexity in HelloWorld3 demonstrates a more sophisticated use of choreography programming languages. It emphasises the language's ability to handle intricate, multi-agent workflows, showcasing its potential for elaborating distributed computing tasks.

Source Code: Deployment and Management Considerations

The following diagram represents a classic Remote Procedure Call (RPC) setup rather than a choreography. This setup differs from HelloWorld3's choreographic approach, although both achieve the same outcome.

Diagram 9: Code without choregraphies, scattered in these three agents

In the diagram, each agent exposes specific functions that can be remotely invoked, demonstrating a more traditional RPC model where the caller dictates the control flow. It is to be excited that the requestor (Agent 1) initiates the process, invoking a function on Agent 2, which invokes a function on Agent 3. After processing, the result travels back through Agent 2 to Agent 1. This is a sequential and direct invocation pattern, with each agent having predefined roles and functions.

Contrastingly, HelloWorld3 describes a choreography where the control flow is described centrally in the choreography description, but it is not centrally executed. The choreography execution emerges from the interactions between the agents. Each agent knows when and where to send the message next, creating a decentralised execution flow.

From a deployment standpoint, the RPC model means that any change in the process logic may require updating the corresponding functions on the respective agents. In the choreography model, updates to the process logic could be managed more fluidly by altering the choreography itself without necessarily altering the individual agents. This enables a mobile code approach in choreographies implemented using swarm communication, providing flexibility and agility in updating the system to reflect changes in business logic. The entire communication protocol can be understood and modified from a single choreography file, increasing transparency and ease of updates, as opposed to digging into the codebase of each agent.

In the context of AI agents, such as those running language or machine learning models, the choreography model allows for the dynamic use of these models based on the instructions received from the choreography rather than having static, hardcoded interactions. This can make the system more responsive to changes and reduce the complexity of the agents, as they rely on the choreography for their behaviour rather than embedded logic.

Swarm Communication: Building and Improving OOP Concepts

As it can be discerned from the examples above when discussing the implementation of executable choreographies using "swarm communication," the actual code of the swarms, referred to in the diagram below as "description of the swarm communications," behaves quite similarly to a class in Object-Oriented Programming (OOP). Therefore, the description of an executable choreography could be considered an evolution of the class concept.

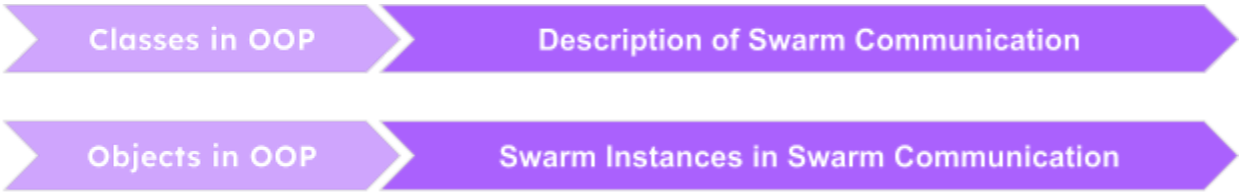

Diagram 10: Similarity with OOP concepts in Swarm Communication

Additionally, objects in OOP can be seen as a specific case of instantiating an “execution flow”.

In OOP, we observe two favours of classes: utility classes that offer functionalities and states to other objects and "flow" classes that compose these utility classes. This observation could be significant for differentiating between code frequently changing due to business logic changes and stable code offering utility to the business logic classes. Unfortunately, this insight has not permeated the creators of programming languages.

Diagram 11: Two code flavours even in OOP

We believe that significant amounts of poor-quality code could be avoided if languages had features to allow the creation of explicitly signalled "choreography" class flavours that document what is possible or not in the composition of other classes. In our examples with agents, the advantage of choreography becomes even more evident because, in the case of RPC, it is not at all obvious which sequence of calls is valid or possible, allowing, in the case of security attacks or causing programming errors, incorrect data, etc.

Indeed, any complex system composed of multiple components inevitably requires the establishment of a communication protocol or usage pattern among and for these components. Regrettably, Object-Oriented Programming (OOP) does not seem to have evolved to incorporate a concept akin to choreography at the level of mainstream programming languages. When it comes to network communication, the situation is similarly peculiar. Due to APIs' simplicity and ease of use, this simplistic approach has prevailed over other initiatives like Enterprise Service Bus (ESB) or executable choreographies, which are rarely used.

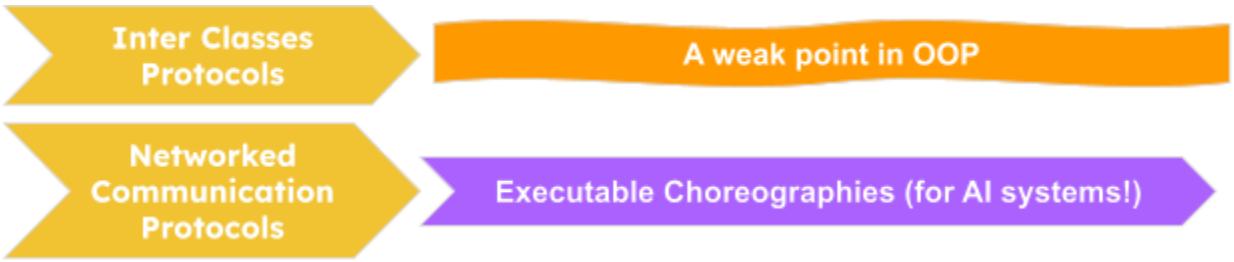

Diagram 12: Programming is about protocols (choreographies)

However, when discussing multi-agent systems or, more generally, more flexible intelligent systems based on natural language, the API approach may encounter an insurmountable barrier. In such scenarios, the methodology of executable choreographies will likely become a necessity. This is due to such systems' dynamic and unpredictable nature, where the interactions between components are not just method calls but complex conversations and negotiations. Executable Choreographies offer a more nuanced and flexible way to model these interactions, documenting the possible sequences of communication and providing a higher level of abstraction that can manage the complexity inherent in intelligent systems. The shift towards choreographies in intelligent systems is seen as a technical necessity and an advancement in design philosophy that aligns better with these systems' organic and evolving nature.

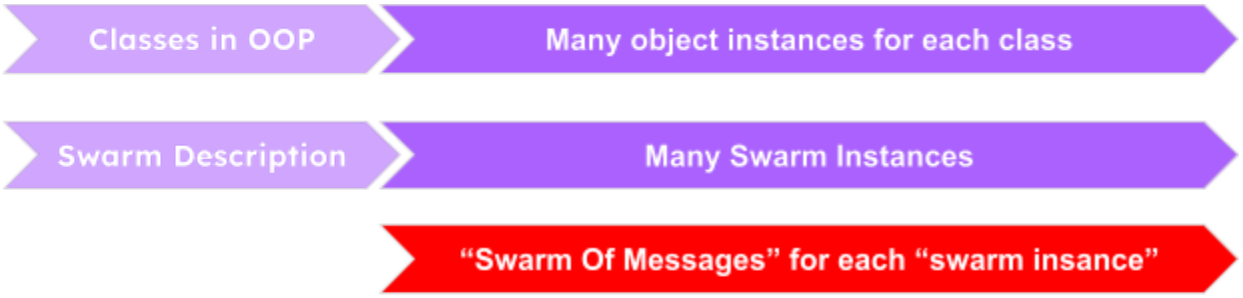

Diagram 13: Each "Swarm instance" has a "swarm of messages"

Some subtleties related to the semantics of "swarm communication" models can be highlighted by how instances within a swarm description function both as "swarms of messages" but also as messages being instances participating in a specific "swarm". By elements of a swarm, we essentially refer to executing a certain phase in an agent, execution caused by a "message" caring the state of the previous instance, also called the "parent" instance. Each method or phase of the swarm has a different "this" in each agent, signifying a local instance in the agent. Various extensions and mechanisms can be envisioned to ensure a common identity for all local "instances" within a swarm execution. Still, it's sufficient to understand that at a conceptual level, this identity is provided because the state of each "child" instance is based on the state of a "parent" instance, which is modified step by step in execution.

It's also clear why there are instances at the "entire swarm" level; because from a start node (as in the examples, the "requester" node), it's possible to start execution flows with entirely different data, which can also vary in the number of swarm-level instances or the order of calling, which, as can be understood from the examples, is dynamically controllable from the code in the "swarm description" phases. Therefore, each "starter" node instantiates a new "swarm of messages". Everything might sound more complex than it is. When you start programming with these concepts, things quickly fall into place. However, this philosophical sophistication can initially seem too abstract or intimidating for some programmers.

Privacy and Security Considerations

From the privacy and security point of view, the best practice of swarm communication is to put the results of computations within the flow while deliberately avoiding transferring confidential data from one node to another. This could involve trust issues about the execution, but digital signatures could help. It is simple but not a very common concept because most use cases are straightforward, and RPC, with eventually some orchestration, is enough. However, with the advent of Large Language Models (LLMs), executable choreographies could become essential to future AI systems architectures. This is because they facilitate proper isolation of each agent's execution environment and enable the verification of communication between agents while maintaining the decentralised nature of the agent swarm.

Diagram 14: The Good, Bad and Ugly strategies when working with Generative AI based Agents

An example that clearly illustrates this point is that with the increasing intelligence of LLMs (Large Language Models), applications will begin incorporating "strings" that represent instructions (prompts) for LLMs. The most appropriate place for these strings is within choreographies rather than scattered throughout the application's code. From a security standpoint, it is sometimes necessary for agents to be highly restricted in their responses, or they might have a role-based system that allows different types of agents to respond differently, or not at all, to unauthorised questions or unauthorised agents. However, because of the LLMs' malleability, embedding all possible questions (prompts) in the API source code looks impractical and is therefore marked as “ugly” in the above diagram. Also, trusting the agent itself is not a very good idea, as an APT attack could be implemented to avoid the hardcoded verifications an attacker could introduce in agents. The proper solution is requesting something like executable choreographies to decouple the prompts in the flow description and allow static and dynamic verification of the prompts. An important concept is that of 'verifiable choreographies’. Static or dynamic checking of the choreographies becomes more realistic to implement, as it does not involve checking the whole code but only the communication protocol itself. The idea is that each agent offers generic APIs and capabilities. Still, the business logic stays at the level of the choreography, offering a decoupling from the long-term code and the ease of changing the business code. Although choreographies inherently promote decentralisation in execution and peer-to-peer (P2P) communication among agents, centralised verification techniques can be implemented transparently, primarily in the communication layer. This layer typically involves message queues for each agent. Still, it can also be more formally established when the choreography is approved by or for each agent. Other verification techniques could involve cryptographic methods and more advanced technical approaches, such as Zero Knowledge Proof or self-validating data [8], where each agent digitally signs its contributions or ensures specific characteristics related to computational integrity.

OpenDSU for AI

Axiologic Research is a major contributor and leader of the OpenDSU open-source project [6]. In this report, we aim to present the underlying principles of the key concepts: KeySSI, DSU, and SVD. We will use examples to illustrate how future AI agents could benefit from these technologies.

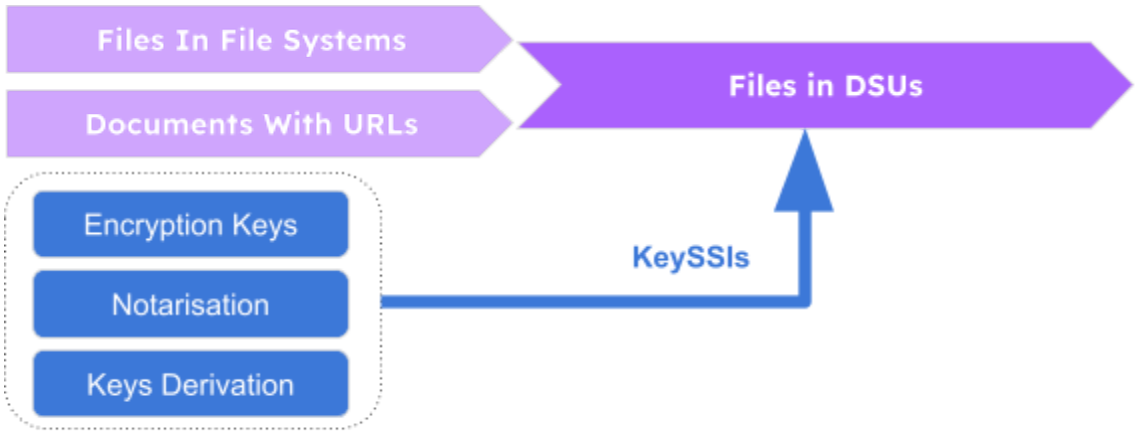

Diagram 15: Potential to evolve the 'file' concept

To understand the concept of DSU, we recommend the documentation available on the OpenDSU website [7]. However, it's crucial to grasp that the fundamental ambition and achievement of OpenDSU is to generalise the idea of a file within a file system by offering the possibility for each file to have a resolvable URL at the internet level. To ensure security, every file is encrypted and optionally notarised in a distributed ledger or blockchain. Notarisation is optional, but when enabled, it transparently assures the programmer of the traceability of all changes and guarantees the immutability of the content and versions of the files. Essentially, we can conceive of the DSU (Data Sharing Unit) as a file system (a key-value database where keys are hierarchical paths) encrypted using encryption keys that can be securely derived to also create a name for the DSU, in combination with the name of the ledger where notarisation occurs. In OpenDSU, we refer to notarisation as anchoring, and the series of derived keys combined with the ledger's name and other derived information that ensures the verification of digital signatures for traceability is called KeySSI (Key + Self Sovereign Identifier).

Diagram 16: A KeySSIs is a URI for encrypted resources

KeySSIs can also provide read-only or write access to these mini file systems called DSUs through specific mechanisms for deriving private keys and combining them with notarisation. The intuition behind this is that even if you do not have access to the key that allows anchoring (based on signing), it is still possible to access a derived key that allows decryption. From what we might call a "top KeySSI" obtained from an initial secret, all other derived keys can be generated, including a key that functions as an identifier for the "DSU anchor" and can be saved in a ledger as the access key to all versions.

Hopefully, this brief presentation has provided an initial understanding of how OpenDSU offers an innovative solution to the challenges of decentralised data sharing by leveraging distributed ledgers, such as blockchains. These ledgers or blockchains necessitate a naming space, and borrowing from the DNS concept, OpenDSU introduces BDNS (Blockchain Domain Naming System). In scenarios where an entity has complete control over data and wishes to securely share it, mostly in read-only mode, such entities or agents will manage a "domain" for a single ledger. This approach facilitates proper data segregation. For multiple stakeholders requiring write access, the ledger must be distributed or blockchain. This latter approach suits NFTs, anchoring "digital twins" or "business processes."

An important insight is that in complex multi-agent environments, each agent will have access only to specific DSUs they work on, allowing choreographies to transmit complex data without transporting big messages, thereby providing security by design and privacy by design. A centralised system used for sharing is an obvious target for attacks, but if a cryptographic key controls sharing, the attack surface is significantly reduced.

If a "business process" involves multiple stakeholders, meaning several agents can modify the state of the business process, it becomes crucial to ensure security and accountability that each agent tracks all changes. Furthermore, the agent responsible should sign each change in the business process. This practice helps maintain a transparent and verifiable record of who made what changes and when enhancing the overall trust and integrity of the process.

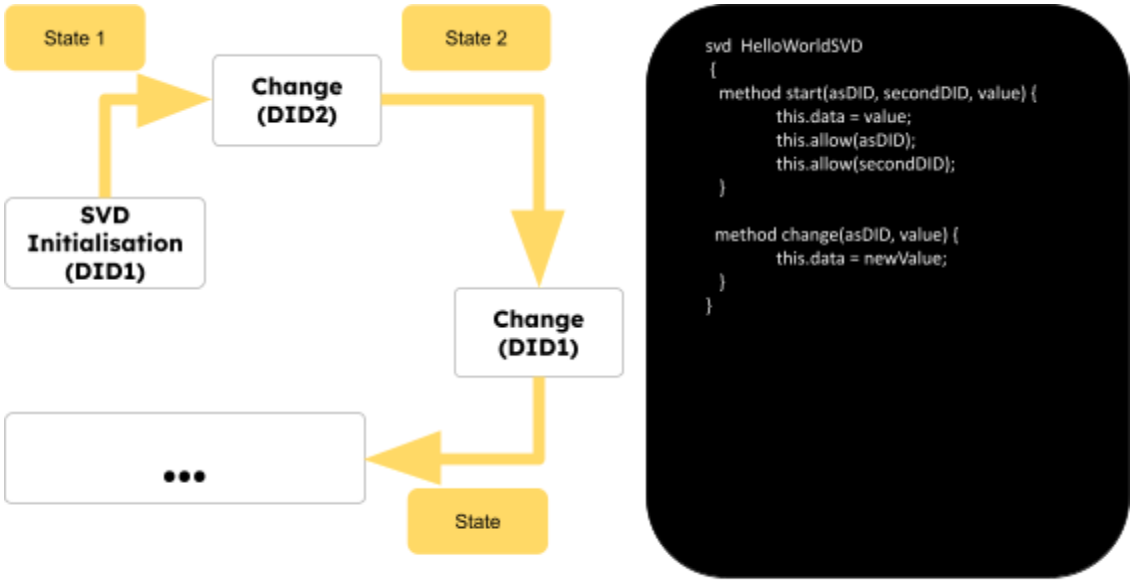

Diagram 17: Simple Example of SVD as micro-ledger smart contract

The concept of Self-Validating Data (SVD) is comparable to the functionality of "smart contracts" stored within a micro-ledger. Essentially, these microledgers could be DSUs or simply a file documenting the progression of a single "smart contract." We have illustrated a simple "Hello World" example in the diagram above to enhance understanding. In this scenario, the start method initialises the process with specific data and sets permissions for two agents. The change method allows these agents to update the process's state, ensuring each modification is accurately recorded and attributed. These agents are identified by Decentralized Identifiers (DIDs). Though the specifics of these identifiers are not very important for this discussion, similar cryptographic properties with W3C DIDs are assumed.

An SVD is conceived as being quite similar to a class; however, each method invocation necessitates an agent's digital signature. The entire history of the class state's evolution is maintained and can be verified at any time. Moreover, the current state can be validated, just as the states of traditional smart contracts can.

SVDs: connecting back to Choreographies

In OpenDSU, the SVDs were imagined as stored in a micro-ledger as the unique smart contract of a single micro-ledger. However, in recent months, we observed that we could connect to the choreographies research, and it could enhance multi-agent alignment properties. Although the following ideas have not yet been implemented in code, it's clear that Self-Validating Data (SVDs) have great potential to offer a robust validation mechanism for the evolution of executable choreographies based on swarm communication. Analogous to SVDs, each step in executing choreographies could be accompanied by digital signatures and the validation of previous states, preventing attackers from deviating from the executable choreography's parameters. This approach integrates the decentralisation, security, and flexibility inherent in executable choreographies with the trust and integrity verification provided by SVDs. Ensuring that each action within the choreography is authenticated and its state transitions are verifiable creates a secure, resilient framework for managing complex interactions across distributed systems. This model enhances the concept of executable choreographies by embedding a mechanism for dynamic adaptation and secure data management, reflecting a sophisticated blend of distributed computing principles with advanced security measures.

Conclusions

The concepts of "executable choreography" based on "swarm communication," DSU as a decentralised data-sharing method, and SVD as a method to implement microledgers are fundamental outcomes of our research over the past decades. They have the potential to become elementary concepts taught in university courses. The complexity introduced by multi-agent systems makes us confident that these approaches are extremely necessary to create proper AI systems in terms of security and privacy. As we advance into an era where AI systems increasingly become part of our daily lives, the importance of these foundational concepts cannot be overstated. They offer a blueprint for constructing AI systems that are efficient, scalable, secure, and respectful of user privacy. By incorporating executable choreographies, DSUs, and SVDs into the fabric of AI systems, we pave the way for a future where these systems can communicate, share data, and make decisions in a manner that is transparent, verifiable, and aligned with ethical standards. This approach significantly enhances the trustworthiness of AI systems and their acceptance by the public and professionals alike. Furthermore, integrating these concepts into educational curricula will equip the next generation of computer scientists and engineers with the tools they need to design and implement AI systems that are inherently secure and privacy-preserving. Ultimately, our research underscores the critical role of advanced cryptographic and decentralised techniques in the ongoing evolution of AI, suggesting a path forward that balances innovation with the imperative to protect and respect individual privacy and security.

References

[IR1] AI Market Evolution report (2023) https://www.axiologic.net/downloads/report_ai_market.pdf
[IR2] Internal Report Super-intelligence Classification (2023)
https://www.axiologic.net/downloads/report_super_intelligences.pdf
[IR3] An internal report on the Social Impact of Controllable AIs (Aligned AIs)
https://www.axiologic.net/downloads/report_social_impact.pdf
[IR4] AssistOS Initial Vision https://www.axiologic.net/downloads/report_assistos.pdf
[IR5] AI Alignment Research Report (2024)
https://www.axiologic.net/downloads/report_ai_alignment.pdf
[1] https://en.wikipedia.org/wiki/Active_message
[2] "Swarm Communication – a Messaging Pattern Proposal for Dynamic Scalability in Cloud"
L. Alboaie, S. Alboaie, P. Andrei, At 15th IEEE International Conference on High-Performance Computing and Communications (HPCC 2013). Zhangjiajie, China, November 2013.
[3] Extending swarm communication to unify choreography and long-lived processes
L. Alboaie, S. Alboaie, T. Barbu, 3rd International Conference on Information Systems Development (ISD2014 Croatia) pp. 375–382, 2014.
[4] "Levels of Privacy for e-Health Systems in the Cloud era" S. Alboaie, L. Alboaie, A. Panu, 24th International Conference on Information Systems Development. Harbin, China, August 25-27, 2015.
[5] Consider checking the www.axiologic.net/research page that provides additional reports and details on the AssistsOS components and concepts.
[6] OpenDSU Documentation (www.opendsu.org)
[7] DSU Concept https://www.opendsu.org/pages/concepts/DSU-Introduction-(RFC-001).html
[8] SVD Concept https://www.opendsu.org/pages/contributors/Self-Validating-Data-(RFC-036).html