The Web 7.0 Architecture Whitepaper (#web7, @web7arch) DIDComm Agent Architecture Reference Model (DIDComm-ARM)

A Design Guide for Software Architects and Developers

Version 0.40 – December 18, 2022

Michael Herman
Self-Sovereign Blockchain
Futurist, Architect, and Developer

Trusted Digital Web
Hyperonomy Digital Identity Lab
Parallelspace Corporation

Alberta, Canada

mwherman@parallelspace.net

This whitepaper is neither a W3C, DIF, Sovrin Foundation, nor ToIP

publication - unofficial, official, or otherwise.

TABLE OF CONTENTS

TABLE OF FIGURES

CHANGE HISTORY

Version 0.1

1. First draft (November 24, 2022)

Version 0.20

1. First complete review draft

Version 0.24

1. Second complete review draft (December 4, 2022)
2. Added Layer 5 DIDComm User Agent Model
3. Added Layer 6 Web 7.0 DIDComm Agent Architecture Model

Version 0.25

1. First public release (December 7, 2022)
2. Added First Principles Thinking reference
3. Added Authentic Conversations Spectrum supporting figure
4. Added a Keystore to each agent in the Layer 6 Web 7.0 DIDComm Agent Architecture Model
5. Added minimum viable mesh network discussion to Layer 4 DIDComm Agent Mesh Network Model

Version 0.27

1. First update (December 11, 2022)
2. Updated DIDComm Messages vs. Verifiable Credentials section in the Layer 3 DIDComm Agent with Verifiable Credential Attachments Model.
3. Updated APPENDIX B – Layer 6 Web 7.0 DIDComm Agent Architecture Model to differentiate the Web 7.0 and the DIDComm Networks. Added positioning for the Internet vs. Web 2.0/3.0 vs Web 7.0. Updated DIDComm Notation model for Layer 6.
4. Started implementing the code examples mentioned under Future Work in the Conclusions section (published here: https://github.com/mwherman2000/web7-didcomm-arm-samples).

Version 0.28

1. Support for multiple DIDComm Networks running on top of multiple network transport protocols.

Version 0.40

1. Personification
2. Several clarifications
3. Updated figures
4. Alignment with DIDComm-ARM PowerPoint presentation deck

ABSTRACT

The purpose of this document is to introduce and describe the DIDComm Agent Architecture Reference Model (DIDComm-ARM) and the DIDComm Notation. The DIDComm-ARM is a layered architecture reference model designed to help guide software architects and developers in the design and creation of the broadest range of DIDComm Agent-based software systems possible. DIDComm Notation is a graphical modeling language used to design and model DIDComm Agent-based software systems.

The goals for this document are three-fold:

- Better understanding of the active components of DIDComm Agent-based software systems and how these components rely on and interact with each other,
- Introduce a new graphical modeling language, DIDComm Notation, to aid architects and developers in visualizing new architectures and designs for DIDComm Agent-based software systems, and
- Describe a layered architecture reference model, DIDComm-ARM, to help guide the design and creation of the broadest range of DIDComm Agent-based software systems possible.

DIDComm Notation contains elements for modeling:

0. Conventional REST/HTTP clients, agents, and services
1. DID Addressable REST/HTTP clients, agents, and services
2. DIDComm clients, agents, and services
3. DIDComm agents that utilize Verifiable Credential message attachments
4. DIDComm mesh networks
5. DIDComm user agents
6. Virtual web drives and keystores

Collectively, these categories of DIDComm Notation modeling elements (and their interrelationships) define the DIDComm Agent Architecture Reference Model (DIDComm-ARM). The DIDComm-ARM contains multiple layered architecture models:

- Layer 0 REST/HTTP Agent Model
- Layer 1 DID Addressable REST/HTTP Agent Model
- Layer 2 DIDComm Agent Model
- Layer 3 DIDComm Agent with Verifiable Credential Attachments Model
- Layer 4 DIDComm Agent Mesh Network Model
- Layer 5 DIDComm User Agent Model (Appendix A)
- Layer 6 Web 7.0 DIDComm Agent Architecture Model (Appendix B)

The intended audience for this whitepaper is a broad range of professionals interested in furthering their understanding of DIDComm for use in software apps, agents, and services. This includes software architects, application developers, and user experience (UX) specialists; as well as people involved in a broad range of standards efforts related to decentralized identity, Verifiable Credentials, and secure storage.

The primary audience is software architects and developers designing and creating DIDComm Agent-based software systems[1].

This document is an independent work product produced by the author(s) and is neither a W3C, DIF, Sovrin Foundation, nor a ToIP work product - unofficial, official, or otherwise.

[1] At the time this document was being conceived (Winter 2022), few, if any, resources were able that focused on the needs of software architects and developers trying to design and create DIDComm Agent-based software systems.

CONTEXT

Purpose

The purpose of this document is to introduce and describe the DIDComm Agent Architecture Reference Model (DIDComm-ARM) and the DIDComm Notation. The DIDComm-ARM is a layered architecture reference model designed to help guide software architects and developers in the design and creation of the broadest range of DIDComm Agent-based software systems possible. DIDComm Notation is a graphical modeling language used to design and model DIDComm Agent-based software systems.

DIDComm Notation contains elements for modeling:

0. Conventional REST/HTTP clients, agents, and services
1. DID Addressable REST/HTTP clients, agents, and services
2. DIDComm clients, agents, and services
3. DIDComm agents that utilize Verifiable Credential message attachments
4. DIDComm mesh networks
5. DIDComm user agents
6. Virtual web drives and keystores

Collectively, the categories of DIDComm Notation modeling elements (and their interrelationships) define the DIDComm Agent Architecture Reference Model (DIDComm-ARM).

This document (the DIDComm-ARM specifically) is an example of First Principles Thinking[2] applied to the concepts of decentralized identifiers, verifiable credentials, and decentralized identifier-based communications.

Goals

This document was produced to address the following 3 goals:

- Better understanding of the active components of DIDComm Agent-based software systems and how these components rely on and interact with each other,
- Introduce a new graphical modeling language, DIDComm Notation, to aid architects and developers in visualizing new architectures and designs for DIDComm Agent-based software systems, and
- Describe a layered architecture reference model, DIDComm-ARM, to help guide the design and creation of the broadest range of DIDComm Agent-based software systems possible.

Intended Audience

The intended audience for this whitepaper is a broad range of professionals interested in furthering their understanding of decentralized identity concepts for use in software apps, agents, and services. This

[2] “Sometimes called “reasoning from first principles,” the idea is to break down complicated problems into basic elements and then reassemble them from the ground up. It’s one of the best ways to learn to think for yourself, unlock your creative potential, and move from linear to non-linear results.”
First Principles: The Building Blocks of True Knowledge (https://fs.blog/2018/04/first-principles/)

includes software architects, application developers, and user experience (UX) specialists; as well as people involved in a broad range of standards efforts related to decentralized identity, Verifiable Credentials, and secure storage.

Organization

This whitepaper contains the following sections:

- Context (this section)
- DIDComm Notation
- Key Actors
- DIDComm Architecture Reference Model (DIDComm-ARM)
- Layer 0 REST/HTTP Agent Model
- Layer 1 DID Addressable REST/HTTP Agent Model
- Layer 2 DIDComm Agent Model
- Layer 3 DIDComm Agent with Verifiable Credential Attachments Model
- Layer 4 DIDComm Agent Mesh Network Model
- Conclusions
- APPENDIX A – Layer 5 DIDComm User Agent Model
- APPENDIX B – Layer 6 Web 7.0 DIDComm Agent Architecture Model
- APPENDIX C – About the Author
- APPENDIX D – Licenses

NOTE: Two late additions to the current version of the DIDComm-ARM include:

- Layer 5 DIDComm User Agent Model
- Layer 6 Web 7.0 DIDComm Agent Architecture Model

NOTE: The descriptions of Layer 5 and Layer 6 are not included in the main body of this whitepaper. A description of the Layer 5 DIDComm User Agent Model has been added as APPENDIX A – Layer 5 DIDComm User Agent Model on page 64. A description of the Layer 6 Web 7.0 DIDComm Agent Architecture Model has been added as APPENDIX B – Layer 6 on page 69.

About the Author

The author is a seasoned software professional with more than 45 years of experience working for software companies large and small; including Microsoft and IBM. His experience includes working as an enterprise architect certified in the use of the ArchiMate enterprise architecture modeling language and the TOGAF enterprise architecture framework. As a professional writer, the author specializes in making new technologies "easier to learn, easier to understand, and easier for you to explain to others".

The author has more than 20 years of experience architecting and implementing decentralized software solutions beginning with the Groove Workspace platform from Groove Networks, Inc. acquired by Microsoft Corporation in 2005 (https://en.wikipedia.org/wiki/Groove_Networks.

In his work with the decentralized identity community, the author is a named contributor to the Decentralized Identifiers (DIDs) v1.0 W3C Recommendation (https://www.w3.org/TR/did-core/#acknowledgements). More details can be found in APPENDIX C – About the Author on page 74.

DIDCOMM NOTATION

DIDComm Notation is a graphical modeling language for designing and visualizing DIDComm Agent-based software systems. DIDComm Notation is focused on the needs of software architects and developers who have a need or desire to design and create DIDComm Agent-based software systems. DIDComm Agent-based software systems, in turn, are comprised of a networked graph of cooperating software clients, agents, and supporting services that exchange application and platform-level messages based on the DIDComm Messaging specification.

DIDComm Messaging

DIDComm Messaging (DIDComm) is a message-based communication framework for software agents to exchange messages in a secure, private, and authenticated manner. DIDComm Messaging builds on top of the capabilities provides enabled by Decentralized Identifiers (DIDs) and DID Registry services. DIDComm Messaging enables a Subject (identified by its associated DID) and its software agent to discover, connect with, and interact with software agents associated with other subjects (these subjects also being identified by their respective DIDs).

The Internet address of a subject's software agent is discovered by looking up the subject's DID in a DID Registry service. A DID Registry service can return the Internet address of the service endpoint exposed by a subject's agent. In addition, the DID Registry can also return additional non-personally identifying information such as the subject's public key. If a subject's DID is registered in a DID Registry, the DID Registry can return a response for the subject's DID that contains the serialized service endpoint and public key information in a JSON-serialized entity known as a DID Document. There is one DID Document for each of the subject's DIDs. A subject can have one or more DIDs registered in the DID Registry.

A subject can be a person, an institution (e.g. a bank, a government office, a school, and another public institution), a large company, or a small business. In addition, a subject can be any non-fungible entity such as a house, boat, car, or a business document (purchase order, purchase order confirmation, waybill, delivery confirmation, invoice, payment receipt, etc.), etc.

The messages exchanged between 2 subjects' agents are authenticated and encrypted such that only the Receiver Agent can decrypt and then authenticate that the message was signed and encrypted by the Sender Agent.

DIDComm Messaging is transport agnostic. DIDComm Messaging only specifies how messages are to be created, signed, and encrypted. The method used to transport the encrypted messages from one subject's agent to another is determined by the application platform. In addition, the payload contained in a DIDComm message can consist of any serialized data (plaintext or binary). This enables both system-specific and application-specific application protocols to be layered on top of DIDComm Messaging with ease.

DIDComm's sole role is to ensure that DIDComm messages, when accepted by the intended Receiver agent, can only be read and processed by the intended Receiver agent. To learn more about the DIDComm Messaging specification, refer to the following resources:

- DIDComm Messaging v2.x specification (https://identity.foundation/didcomm-messaging/spec/)
- DIDComm.org community website (https://didcomm.org/)
- decentralized-identity / didcomm-messaging GitHub repository (https://github.com/decentralized-identity/didcomm-messaging)

Layered Architecture Reference Model

The layering of the DIDComm Agent Architecture Reference Model, determined how the elements of the DIDComm Notation were partitioned and categorized. At the highest level, the DIDComm Notation elements are partitioned into those required to support:

a. Layer 0 REST/HTTP Agents,
b. Layer 1 DID Addressable REST/HTTP Agents, and
c. Layer 2 (and above) DID Agents.

The following sections introduce and describe the graphical elements of DIDComm Notation, layer by layer. This list of elements includes:

- REST/HTTP Origin and Endpoint Interfaces (Layer 0)
- REST/HTTP Clients, Agents, and Services (Layer 0)
- Decentralized Identifier Registries (DID Registries) (Layer 1)
- DIDComm Origin and Endpoint Interfaces (Layer 2)
- DIDComm Clients and Agents (Layer 2)
- Verifiable Credentials and Verifiable Credential Attachments (Layer 3)

Later in the document, after the above elements have been introduced and put into context, the following elements are described as they are encountered:

- Decentralized Identifiers (DIDs)

DIDComm is strictly a message-passing-based communications framework. DIDComm does not support any sort of Remote Procedure Call (RPC) semantics (see Design APIs for microservices (https://learn.microsoft.com/en-us/azure/architecture/microservices/design/api-design)); rather it relies on asynchronous, one-way message passing to deliver messages from a software component with a communications role of (Message) Sender Agent to a software component with a communications role (Message) Receiver Agent. DIDComm is network transport agnostic meaning that potentially any Internet communications protocol can be used including but not limited to:

- REST/HTTP
- gRPC
- SMTP (yes, even email)
- lib2p2
- QR codes
- Bluetooth

- Message queuing technology and products
- Store-and-forward services
- Content delivery services
- Streaming services
- Web servers
- Business integration services
- Workflow platforms
- Etc.

DIDComm messages can be transported over any Internet transport capable of transporting a reasonably sized payload. DIDComm does not specify any limits on the size of a message payload but a particular agent implementation may stipulate a specific **max_receive_bytes** value (https://identity.foundation/didcomm-messaging/spec/v2.0/#max_receive_bytes).

Key Actors

The key actors in the DIDComm-ARM scenarios that follow are Alice, Bob, ABC Grocery, and David's Cabbages. They are depicted using DIDComm Notation as shown in the following figures.

Alice
(did:person:1234)
Bob
(did:person:5678)

Figure 1. DIDComm Notation: Actors Alice and Bob

Alice

- Alice is permanently assigned or associated with decentralized identifier **did:person:1234**.
- Alice can be assigned other decentralized identifiers but **did:person:1234** is the only identifier used in the scenarios described in the document.
- These additional decentralized identifiers can be used by Alice to manage different personas or roles Alice may have in her life, for example.

Bob

- Bob is permanently assigned or associated with decentralized identifier **did:person:5678**.
- Bob can be assigned other decentralized identifiers but **did:person:5678** is the only identifier used in the scenarios described in the document.
- These additional decentralized identifiers can be used by Bob to manage different personas or roles Bob may have in his life, for example.

NOTE: In the figure above, carefully note that Alice and Bob each have their own decentralized identifiers chosen from a hypothetical DID method space called "did:person". Also, note that "did:person:1234" is used to uniquely identify or refer to Alice, the concrete person (or subject). The DID doesn't, for example, name or refer to Alice's software agent (if Alice has one). Alice's agent is identified or named by its Internet address (which may be as simple as a domain name and port number or a multi-component Internet URL). The mapping from Alice's DID to the Internet address of Alice's software agent is maintained by a DID Registry if Alice is registered with the DID Registry (see section DID Registry on page 18 for more information on DID Registries). Alice's agent and/or the version of the software code that implements Alice's agent may have their own DIDs but this should not be confused with Alice's DID that refers to her as a subject, as a concrete person. Similarly for Bob.

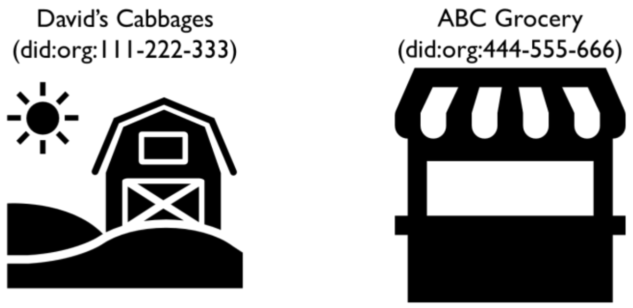


Figure 2. DIDComm Notation: Actors David's Cabbages and ABC Grocery

David's Cabbages

- David's Cabbages is permanently assigned or associated with decentralized identifier **did:org:111-222-333**.
- David's Cabbages can be assigned other decentralized identifiers but **did:org:111-222-333** is the only identifier used in the scenarios described in the document.
- These additional decentralized identifiers can be used by David's Cabbages to manage different business personas or roles David's Cabbages may have, for example.

ABC Grocery

- ABC Grocery is permanently assigned or associated with decentralized identifier **did:org:444-555-666**.
- ABC Grocery can be assigned other decentralized identifiers but **did:org-444-555-666** is the only identifier used in the scenarios described in the document.
- These additional decentralized identifiers can be used by ABC Grocery to manage different business personas or roles ABC Grocery may have, for example.

In the layered DIDComm-ARM model, Layer 0 REST/HTTP defines a foundation layer based on REST/HTTP protocols and no DIDComm artifacts whatsoever. This layer and this approach provide an easy-to-understand foundation on top of which various decentralized identifier and DIDComm capabilities can be easily layered and understood.

Layer 0 REST/HTTP Agent Model Elements

The purpose of the Layer 0 REST/HTTP elements is to enable the design and creation of REST/HTTP Agent-based software systems. REST/HTTP Agent-based systems rely on Clients and Agents communicating by sending HTTP PUT messages from Client applications to Agent applications. A REST/HTTP message payload can be any plain text or serializable data.

REST/HTTP Origin and Endpoint Interfaces

A REST/HTTP Agent has the ability to both send and receive a message payload. A REST/HTTP Client can only send a message payload. There are 3 different types of interfaces defined for REST/HTTP Agents and Clients (as illustrated below).

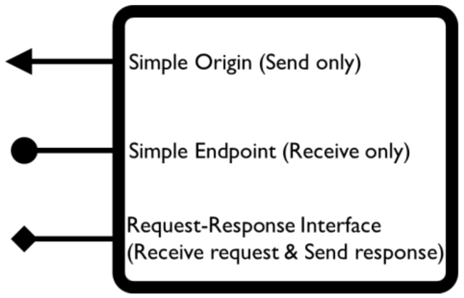

Simple Origin (Send only)
Simple Endpoint (Receive only)
Request-Response Interface
(Receive request & Send response)

Figure 3. DIDComm Notation: REST/HTTP Origin and Endpoint Interfaces

REST/HTTP Origin Interfaces

The component of a REST/HTTP Agent or REST/HTTP Client that can send a message payload is denoted as a line with an arrowhead. This component is called an Origin Interface – it is the Origin Interface within an Agent or Client where new messages are created and sent to a Receiver Agent. No response is expected to be returned by the Receiver Agent (except for an HTTP status code).

REST/HTTP Endpoint Interfaces

The component of a REST/HTTP Agent that can receive a message payload is denoted as a line with a round endpoint. This component is called an Endpoint – it is the Endpoint interface within an Agent or Client where inbound messages are received and accepted from a Sender Agent. No response is returned by the Receiver Agent (except for an HTTP return code).

Request-Response Interfaces

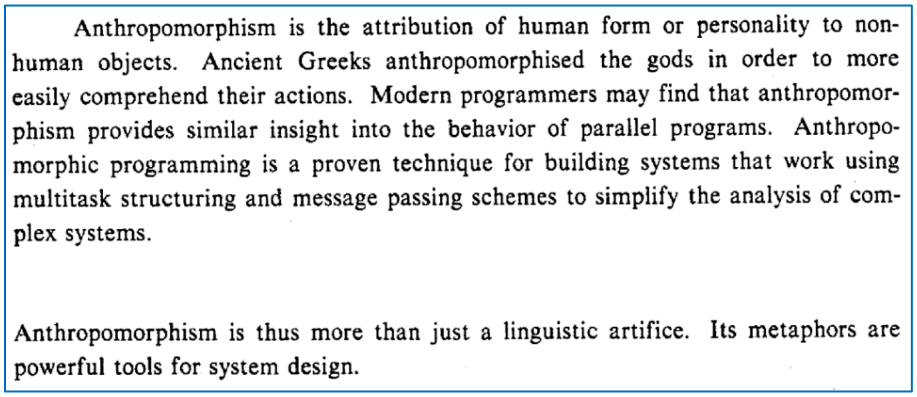
There is a third variation: a Request-Response Interface that may be found on a REST/HTTP Service. This is a traditional HTTP request-response communication pattern where a Service returns a response once it has received, accepted, and processed an inbound request. A Request-Response Interface is denoted as a line with a diamond-shaped endpoint.

Anthropomorphic Programming

The concept of an Agent used throughout this whitepaper was heavily influenced by concepts underlying the discipline of Anthropomorphic Programming. These concepts are as described in the following technical reports from the University of Waterloo:

- Kellogg S. Booth et al., Anthropomorphic Programming (https://cs.uwaterloo.ca/research/tr/1982/CS-82-47.pdf), 1982.

- Darlene A. Plebon et al., Interactive Picture Creation Systems (https://cs.uwaterloo.ca/research/tr/1982/CS-82-46.pdf), 1982.
- David R. Cheriton, Multi-Process Structuring - and the Thoth Operating System (https://cs.uwaterloo.ca/research/tr/1979/CS-79-19.pdf), 1979

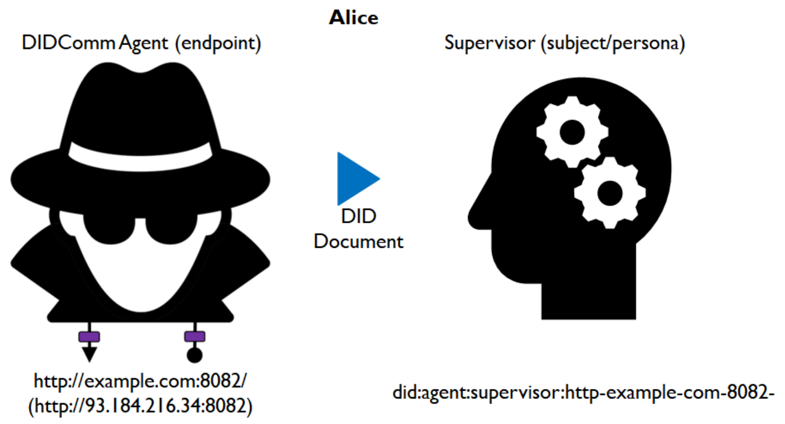


Anthropomorphism is the attribution of human form or personality to non-human objects. Ancient Greeks anthropomorphised the gods in order to more easily comprehend their actions. Modern programmers may find that anthropomorphism provides similar insight into the behavior of parallel programs. Anthropomorphic programming is a proven technique for building systems that work using multitask structuring and message passing schemes to simplify the analysis of complex systems.

Anthropomorphism is thus more than just a linguistic artifice. Its metaphors are powerful tools for system design.

Figure 4. What is Anthropomorphic Programming[3]?

Applied to the concept of REST/HTTP (and DIDComm) Agents, anthropomorphic design (and programming) are powerful tools for characterizing (humanizing) the role and functions implemented by a particular Agent. In the DIDComm-ARM, anthropomorphic design adds a second element next to the REST/HTTP (or DIDComm Agent) to represent (or present an abstraction for) the role and functions provided by the Agent. Here is an example of a DIDComm Agent whose personification is a Supervisor. Personifications are assigned decentralized identifiers. Agents are identified by their service endpoint URLs (protocol, address, port, path). A DID Document (via the DID Registry) enables the mapping from a personification's DID to the associated Agent's service endpoints.


Alice
DIDComm Agent (endpoint)
Supervisor (subject/persona)
DID
Document
http://example.com:8082/
(http://93.184.216.34:8082)
did:agent:supervisor:http-example-com-8082-

To learn more about using anthropomorphic design and personification in the DIDComm-ARM, see section Personification of DIDComm Agents on page 25.

[3] Kellogg S. Booth et al., Anthropomorphic Programming (https://cs.uwaterloo.ca/research/tr/1982/CS-82-47.pdf), 1982.

REST/HTTP Clients, Agents, and Services

There are 3 types of software components in the Layer 0 REST/HTTP Agent in the DIDComm-ARM as illustrated in the figure below. These include:

- REST/HTTP Clients
- REST/HTTP Agents
- REST/HTTP Services

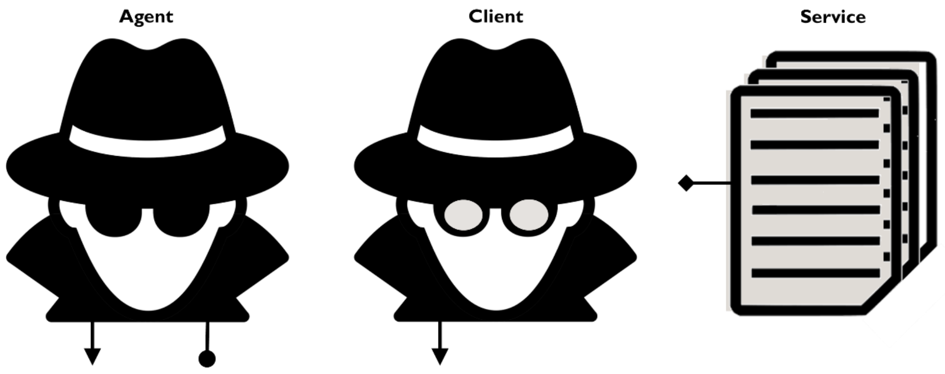


Figure 5. DIDComm Notation: REST/HTTP Agent Model: a) Agent, b) Client, and c) Service

REST/HTTP Clients

The primary role of a REST/HTTP Client is to act as a Sender Agent or initiator of a REST/HTTP message to be received by a Receiver REST/HTTP Agent. In addition, a REST/HTTP Client may initiate a Request-Response message interaction with a Service. A REST/HTTP Client has one interface, an Origin Interface, for creating and sending a plaintext or serializable data payload to a REST/HTTP Agent.

REST/HTTP Agents

The primary roles of a REST/HTTP Agent are to act as both a Sender Agent and Receiver Agent of REST/HTTP messages. In addition, a REST/HTTP Agent may initiate a Request-Response message interaction with a Service. A REST/HTTP Agent has two interfaces, an Origin Interface and Endpoint Interface, for creating and sending a plaintext or serializable data payload to a REST/HTTP Agent and receiving and accepting an inbound message from a REST/HTTP Client or another REST/HTTP Agent.

REST/HTTP Services

A REST/HTTP Service has a single Request-Response interface for receiving an inbound Request message and replying with a Response message.

Layer 1 DID Addressable REST/HTTP Agent Model Elements

To support Layer 1's additional requirement of being able to discover and interact with the Agent associated with a subject (i.e. interact with the Agent associated with a person, institution, company, business document, etc.) via the subject's decentralized identifier, an additional software component needs to be added to the DIDComm-ARM: a Decentralized Identifier Registry (DID Registry).

DID Registry

The purpose of a DID Registry is two-fold:

- to return the Internet address of the service endpoint exposed by a subject's agent, and
- to return additional non-personally identifying information such as the subject's public key.

If a subject's DID is registered in a DID Registry, the DID Registry can return a response for the subject's DID that contains the serialized service endpoint and public key information in a JSON-serialized entity known as a DID Document. There is one DID Document for each of the subject's registered DIDs. A subject can have one or more DIDs registered in the DID Registry.

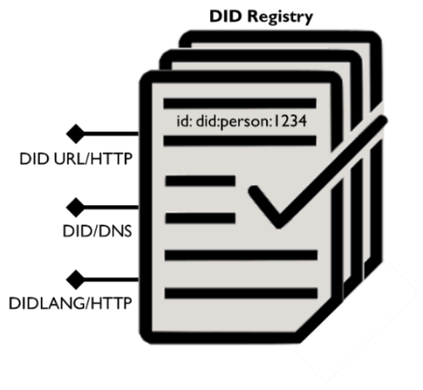


Figure 6. DIDComm Notation: DID Registry Endpoint Interfaces

The above figure shows that a DID Registry can expose its functionality through a range of Request-Response protocols:

- DID URL/HTTP
- DID/DNS
- DIDLANG/HTTP

DID URL/HTTP

DID URL/HTTP corresponds to the classic Decentralized Identifier Resolution (DID Resolution) protocol described here: https://w3c-ccg.github.io/did-resolution/. One implementation can be found here: https://danubetech.com/tech/uni-resolver (including an online, hosted version here: https://godiddy.com/).

DID/DNS

DID/DNS is implemented by adding a range of new DNS Resource Record types to an industry-standard, specifications-compliant Domain Name Service (DNS) Server. One implementation is described here: https://hyperonomy.com/2019/12/03/trusted-digital-web-first-trusted-web-page-delivered-today-dec-3-2019/. The Trusted Digital Web Data Registry can be transparently deployed as a forwarding DNS server (logically in front of any existing DNS service). The open-source repository for this implementation can be found on GitHub: https://github.com/mwherman2000/DnsServer. The DNS (Domain Name Service) specification can be found here: https://www.ietf.org/rfc/rfc1035.txt. A primer on the DNS protocol can be found here: https://hyperonomy.com/2019/01/02/dns-domain-name-service-a-detailed-high-level-overview/.

DIDLANG/HTTP

DIDLANG/HTTP is an experimental domain-specific language (DSL) for supporting full CRUD capabilities against DID Documents in DID Registries and DID Objects managed by DID Agents. Based on the concepts of DID Indirection and DID Coercion, DIDLANG is the equivalent of "SQL for DIDs, DID Documents, and DID Objects". One open-source implementation is described here: https://github.com/mwherman2000/BlueToqueTools#didlang-language-command-line-interpreter-for-did-method-namespaces-did-identifiers-did-documents-did-agent-service-endpoints-did-agent-servers-did-agent-clusters-and-did-objects-version-04. A demonstration of the current DIDLANG processor can found here: https://www.youtube.com/playlist?list=PLU-rWqHm5p45RKAAF8bleTuPNI5cBE3gP.

For this document, it is assumed the DID Registry component only supports the DIDURL/HTTP (aka DID Resolution) protocol as illustrated below. This DID Registry accepts a DID as a single query parameter in the Request message and returns a copy of the DID Document associated with the DID query parameter (if the DID is registered with the DID Registry).

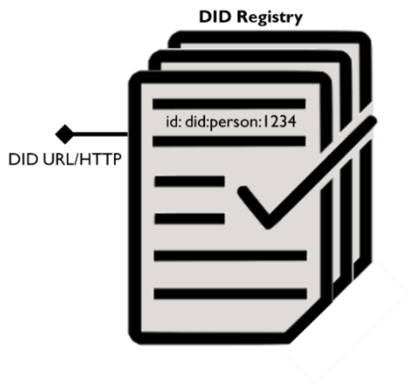


Figure 7. DIDComm Notation: DID Registry with a single DID URL/HTTP Interface

Layer 2 DIDComm Agent Model Elements

The purpose of the Layer 2 DIDComm elements is to enable the design and creation of DIDComm Agent-based software systems. DIDComm Agent-based systems rely on Clients and Agents communicating by sending HTTP PUT messages from Client applications to Agent applications. A DIDComm message payload differs from a Layer 0 REST/HTTP message payload in the fact that the entire payload is designed to be authenticated (signed) and private (encrypted) in such a way that only the intended Receiver Agent can open, decrypt, authenticate, and read the payload.

DIDComm Origin and Endpoint Interfaces

A DIDComm Agent has the ability to both send and receive an authenticated (signed), private (encrypted) message that can only be opened, decrypted, and read by the intended Receiver Agent. A DIDComm Agent Client can only send authenticated and private messages to a Receiver Agent. A DIDComm Agent Client, by definition, has no ability to receive DIDComm Messages. There are 2 different types of interfaces defined for DIDComm Agents and Clients (as illustrated below).

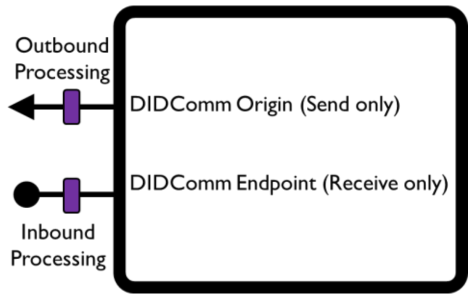


Figure 8. DIDComm Notation: DIDComm Agent Origin and Endpoint Interfaces

DIDComm Origin Interfaces

The component of a DIDComm Agent or DIDComm Client that can send a message payload is denoted as a line with an arrowhead. This component is called an Origin Interface – it is the Origin Interface within an Agent or Client where new messages are created and sent to a Receiver Agent. No response is expected to be returned by the Receiver Agent.

The key difference between a DIDComm Origin Interface and a REST/HTTP Origin Interface is the Outbound Processing that occurs as the last step before the message is sent to the Receiver Agent (as illustrated by the small purple rectangle labeled Outbound Processing in the above diagram). Outbound Processing is responsible for signing the message with the private key of the Sender Agent and encrypting the payload with the public key of the Receiver Agent. This enables the Receiver Agent to verify the authenticity of the Sender Agent as well as ensure the Receiver Agent is the intended recipient (and hence, able to decrypt the message payload).

DIDComm Endpoint Interfaces

The component of a DIDComm Agent that can receive a message payload is denoted as a line with a round endpoint. This component is called an Endpoint – it is the Endpoint interface within an Agent or Client where inbound messages are received and accepted from a Sender Agent. No response is sent by the Receiver Agent.

The key difference between a DIDComm Endpoint and a REST/HTTP Endpoint is the Inbound Processing that occurs as the first step after the message is received and accepted by a Receiver Agent (as illustrated by the small purple rectangle labeled Inbound Processing in the above diagram). Inbound Processing is responsible for decrypting the payload with the Receiver Agent's private key and authenticating the Sender Agent as the agent that originated the message. This enables the Receiver Agent to verify the authenticity of the Sender Agent as well as ensure the Receiver Agent is the intended recipient (and hence, able to decrypt the message payload).

DIDComm Clients and DIDComm Agents

There are 2 types of software components in Layer 2 DIDComm Agents in the DIDComm-ARM as illustrated in the figure below. These include:

- DIDComm Clients
- DIDComm Agents

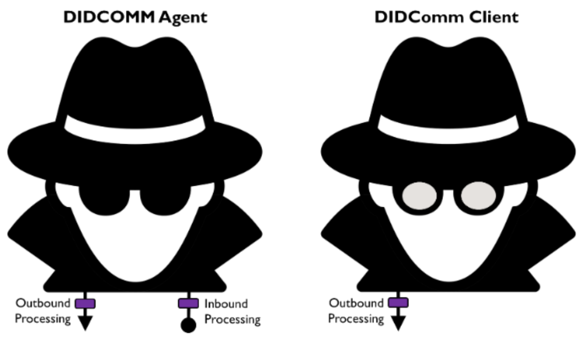


Figure 9. DIDComm Notation: DIDComm Agent Model: a) DIDComm Agent and b) DIDComm Client

DIDComm Clients

The primary role of a DIDComm Client is to act as a Sender Agent or initiator of a DIDComm message to be received by a Receiver Agent DIDComm Agent. In addition, a DIDComm Client may initiate a Request-Response message interaction with a REST/HTTP Service (e.g. DID Registry). A DIDComm Client has one interface, an Origin Interface, for creating and sending authenticated (signed) and private (encrypted) messages to a Receiver Agent DIDComm Agent.

The signing and encryption are performed by the Outbound Processing component (small purple rectangle in the above diagram) as the last step before the message is sent to the Receiver Agent DIDComm Agent.

DIDComm Agents

The primary roles of a DIDComm Agent are to act as both a Sender Agent and Receiver Agent of DIDComm messages. In addition, a DIDComm Agent may initiate a Request-Response message interaction with a REST/HTTP Service (e.g. DID Registry). A DIDComm Agent has two interfaces, an Origin Interface and an Endpoint for creating and sending authenticated (signed) and private (encrypted) messages to a Receiver Agent DIDComm Agent and receiving, accepting, decrypting, and authenticating an inbound message from a DIDComm Client or another DIDComm Agent.

The decrypting and authentication are performed by the Inbound Processing component (small purple rectangle in the above diagram) as the first step after the message is received and accepted by the Receiver Agent DIDComm Agent.

DIDComm User Agents

New capabilities usually require the addition of a corresponding element to the DIDComm Notation. Below is the visual language element symbolizing a DIDComm User Agent with 2 messaging interfaces: a DIDComm Origin Interface and a DIDComm Endpoint Interface.

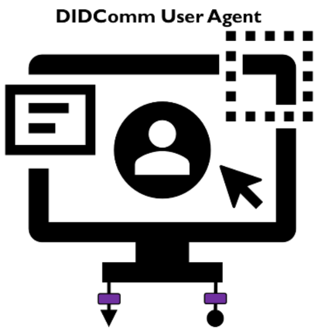


Figure 10. DIDComm Notation: DIDComm User Agent

A DIDComm User Agent projects a user interface for a human actor to use to interact with the decentralized identifier-based software system. The Origin messaging interface has the sole purpose of enabling the DIDComm User Agent application to send DIDComm Messages to other DIDComm Agents. The Endpoint messaging interface's sole purpose is to receive inbound messages – not from an arbitrary DIDComm Agent – but only from the specific DIDComm Agent that it's personally paired with. Any external messages from other DIDComm Agents are received by this latter agent (see below).

Line-of-Business Applications

The DIDComm Notation element symbolizing a (non-DIDComm enabled) line-of-business application appears below.

Figure 11. (Non-DIDComm enabled) Line-of-Business Application

DIDComm Agent Application Gateways

The following is an example of DIDComm Client serving in the role of DIDComm Agent (Client) Application Gateway connected to a legacy (non-DIDComm enabled) line-of-business application (e.g. a healthcare records system).

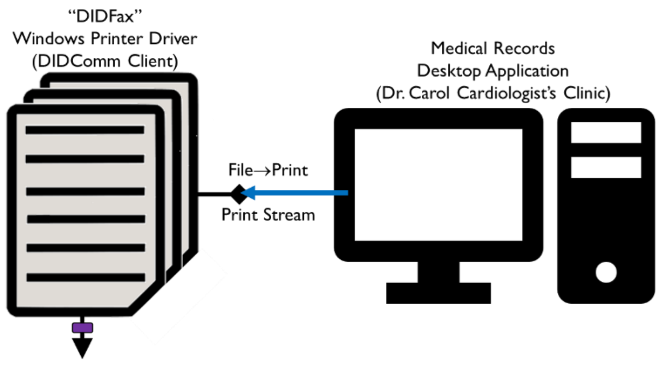
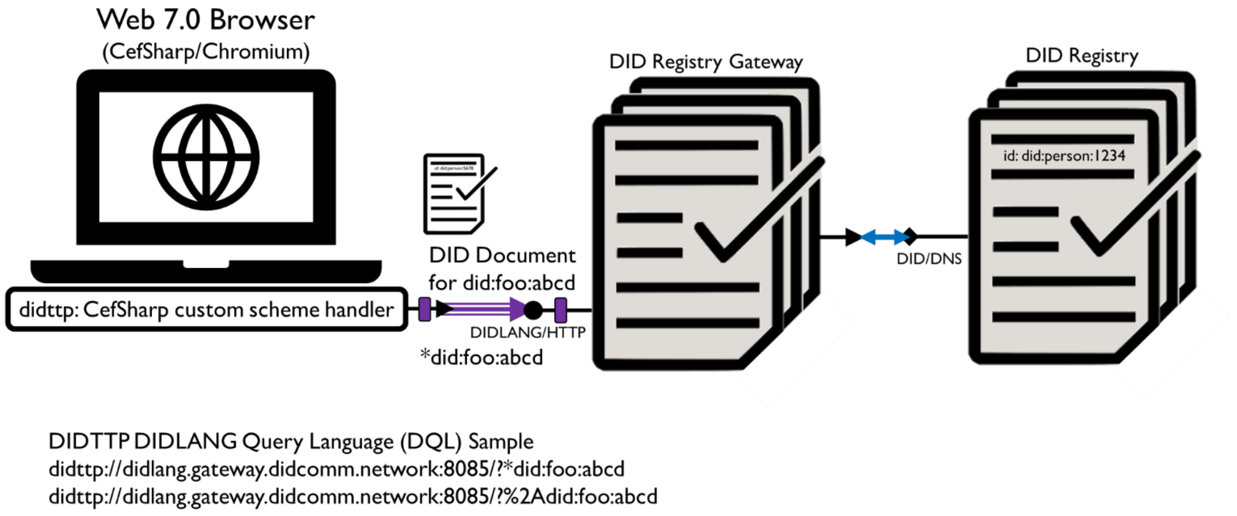


Figure 12. DIDComm Agent Application Gateway Example: Healthcare Records Application Gateway

The following figure depicts 2 types of DIDComm Agent Application Gateways.

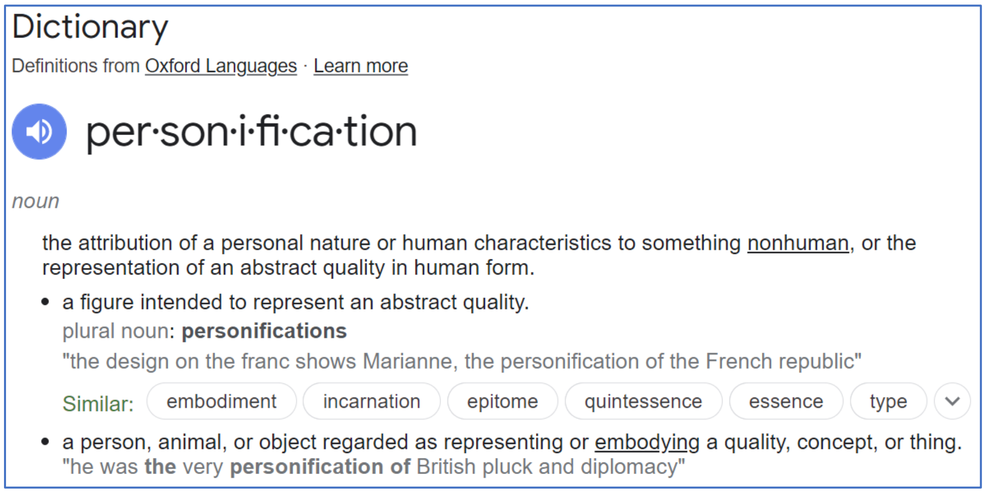


DIDTTP DIDLANG Query Language (DQL) Sample
didttp://didlang.gateway.didcomm.network:8085/?*did:foo:abcd
didttp://didlang.gateway.didcomm.network:8085/?%2Adid:foo:abcd

Figure 13. a) Web 7.0 Browser DIDComm Client Application Gateway, b) DIDComm Agent DID Registry Application Gateway

Personification of DIDComm Agents

The following is a dictionary definition for personification[4]:



Dictionary

Definitions from Oxford Languages · Learn more

per·son·i·fi·ca·tion

noun

the attribution of a personal nature or human characteristics to something nonhuman, or the representation of an abstract quality in human form.

- a figure intended to represent an abstract quality.
  plural noun: **personifications**
  "the design on the franc shows Marianne, the personification of the French republic"
  Similar: embodiment incarnation epitome quintessence essence type
- a person, animal, or object regarded as representing or embodying a quality, concept, or thing.
  "he was **the** very **personification of** British pluck and diplomacy"

Figure 14. Personification: Dictionary Definition

In the context of the DIDComm-ARM, DIDComm Agents, and the design of decentralized identifier-based software systems, there are scenarios where it is sometimes easy to identify where it would be useful to use a DIDComm Agent; for example, as DIDComm Message Router Agent in the DIDComm Network (refer to Layers 4, 5 or 6 of the DIDComm-ARM). Naturally, because the Message Router Agent is a special-purpose DIDComm Agent, it is expected to have a DIDComm Endpoint interface and also a DIDComm Origin interface as shown in the figure below. But how is a neighboring Message Router Agent or a regular DIDComm Agent supposed to be able to discover what the Internet address (and port number) is the Message Router Agent's DIDComm Endpoint interface (aka service endpoint interface)?

DIDComm Agents do not have decentralized identifiers. There appears to be nothing that can be used to look up the Message Router Agent's DID Document in a DID Registry because of this.

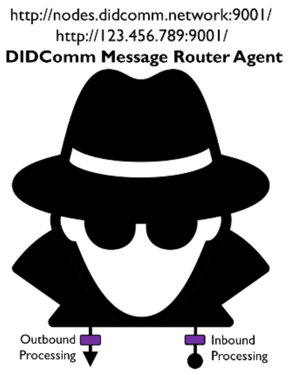

http://nodes.didcomm.network:9001/
http://123.456.789:9001/
DIDComm Message Router Agent
Outbound Processing
Inbound Processing

Figure 15. DIDComm Message Router Agent

This is where *personification* kicks. From the dictionary definition given above:

[4] Google Search

Personification: the attribution of a personal nature or human characteristics to something nonhuman, or the representation of an abstract quality in human form.

The Message Router Agent in the above figure needs to be personified so that its personification can, in turn, be assigned a decentralized identifier (as shown in the following diagram).

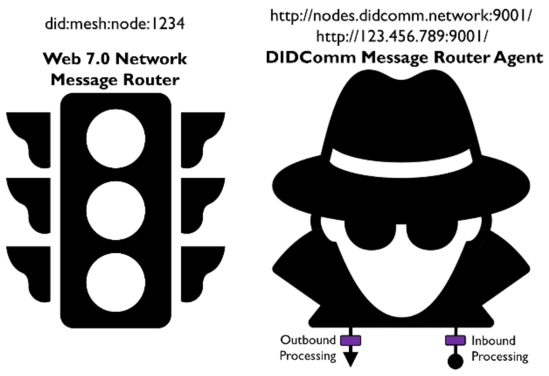

did:mesh:node:1234
Web 7.0 Network Message Router
http://nodes.didcomm.network:9001/
http://123.456.789:9001/
DIDComm Message Router Agent
Outbound Processing
Inbound Processing

Figure 16. DIDComm Message Router Agent and its Personification

The personification, in this case a traffic light, is analogous to a person being assigned as the personification of Alice's or Bob's personal DIDComm Agent. It is the personifications of a DIDComm Agent that can be assigned a decentralized identifier, which, in turn, can be used to retrieve the associated DID Document from the DID Registry where the DID is registered. From this DID Document, the address of the Message Router Agent's DIDComm Endpoint interface can be extracted.

NOTE: In addition, because this is DIDComm Message Router Agent, the DID Document will also contain addresses of all of the Agent's nearest neighbor DIDComm Message Router Agents in the DIDComm Network.

In DIDComm Notation, rather than having to draw each DIDComm Agent's personification as a large icon each time, the following graphical shorthand can be used.

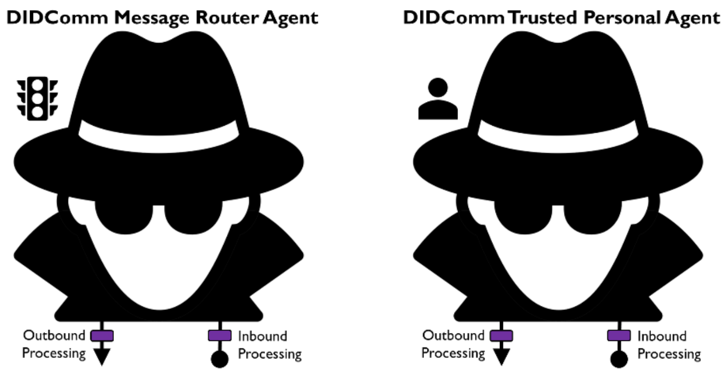


Figure 17. DIDComm Agent Personifications: a) Message Router Agent and b) DIDComm Trust Personal Agent

Layer 3 DIDComm Agent with Verifiable Credential Attachments Model Elements

This section provides a brief overview of what constitutes a Verifiable Credential and outlines how Verifiable Credentials can be attached to a DIDComm Message.

Verifiable Credentials

A Verifiable Credential is depicted by the following element in the DIDComm Notation.

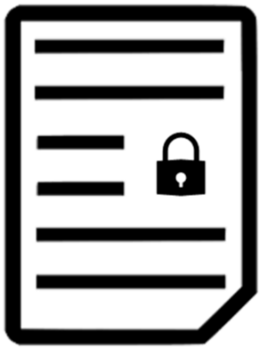


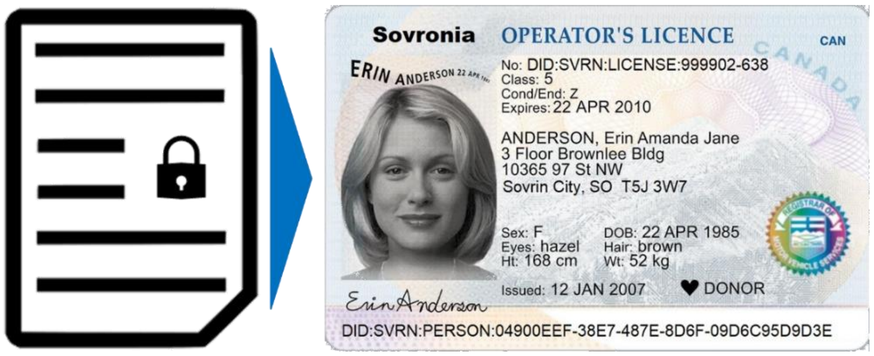
Figure 18. DIDComm Notation: Verifiable Credential

Verifiable Credential Examples

The following are examples of hypothetical Verifiable Credentials and their renderings. These are examples of 3 types of verifiable data that can be serialized into the Verifiable Credential serialization format:

- Erin's driver's license (operator's license)
- Business Card based on the Person schema example from Schema.org
- Purchase order for 10 cabbages

Erin's Driver's License (Operator's License)



Sovronia OPERATOR'S LICENCE CAN

ERIN ANDERSON

No: DID:SVRN:LICENSE:999902-638
Class: 5
Cond/End: Z
Expires: 22 APR 2010

ANDERSON, Erin Amanda Jane
3 Floor Brownlee Bldg
10365 97 St NW
Sovrin City, SO T5J 3W7

Sex: F DOB: 22 APR 1985
Eyes: hazel Hair: brown
Ht: 168 cm Wt: 52 kg

Issued: 12 JAN 2007 ♥ DONOR

Erin Anderson

DID:SVRN:PERSON:04900EEF-38E7-487E-8D6F-09D6C95D9D3E

Figure 19. Rendering of a Driver's License Verifiable Credential

Business Card based on the Person schema example from Schema.org

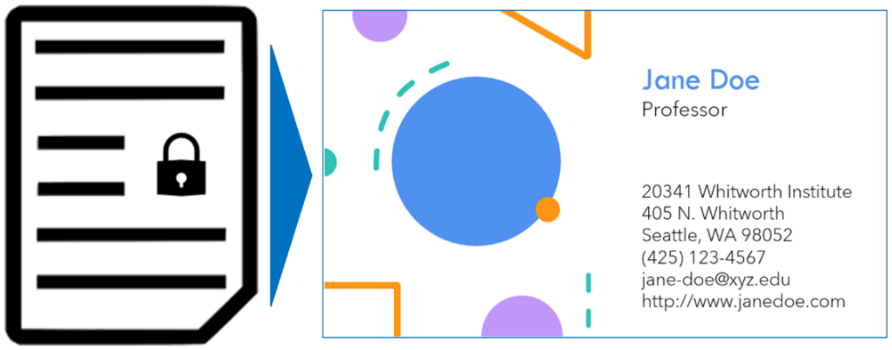
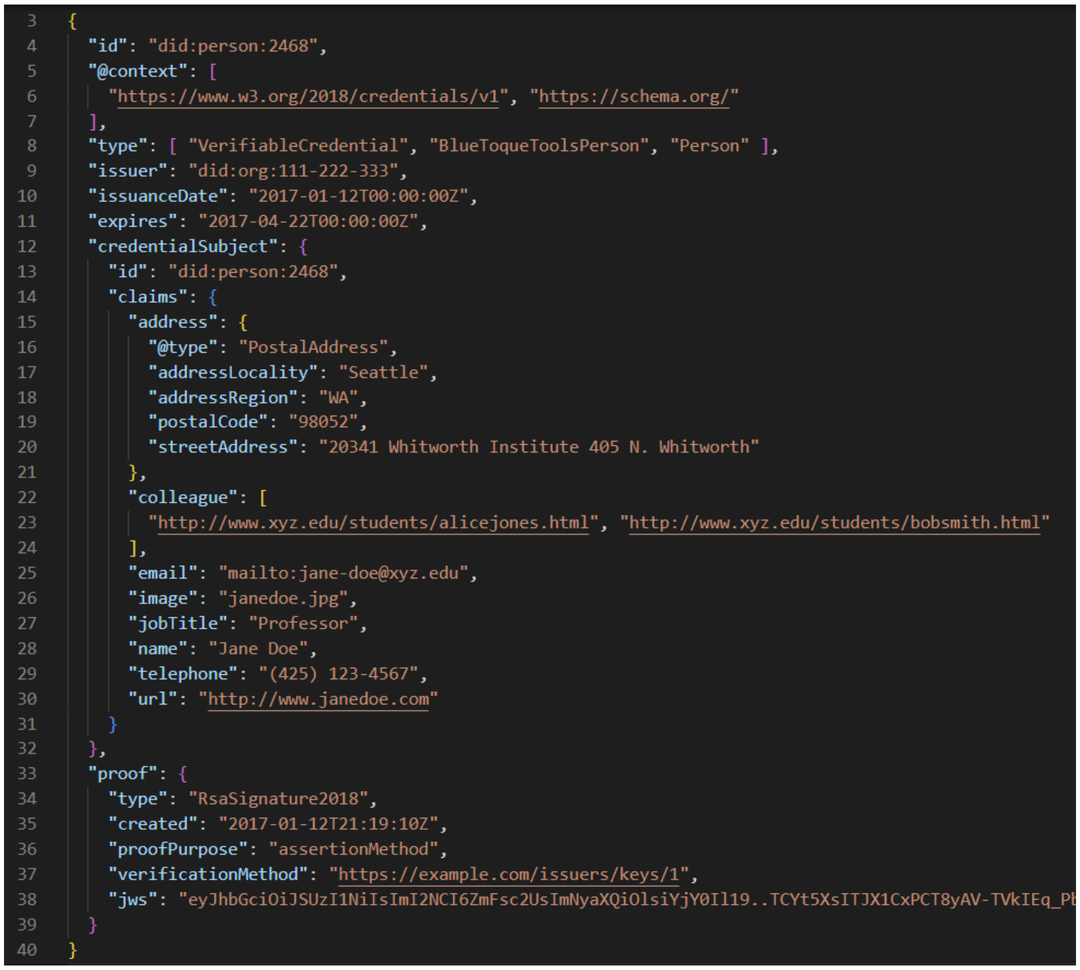


Figure 20. Rendering of a Business Card based on the Person schema example from Schema.org[5]



```
{
  "id": "did:person:2468",
  "@context": [
    "https://www.w3.org/2018/credentials/v1", "https://schema.org/"
  ],
  "type": [ "VerifiableCredential", "BlueToqueToolsPerson", "Person" ],
  "issuer": "did:org:111-222-333",
  "issuanceDate": "2017-01-12T00:00:00Z",
  "expires": "2017-04-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:person:2468",
    "claims": {
      "address": {
        "@type": "PostalAddress",
        "addressLocality": "Seattle",
        "addressRegion": "WA",
        "postalCode": "98052",
        "streetAddress": "20341 Whitworth Institute 405 N. Whitworth"
      },
      "colleague": [
        "http://www.xyz.edu/students/alicejones.html", "http://www.xyz.edu/students/bobsmith.html"
      ],
      "email": "mailto:jane-doe@xyz.edu",
      "image": "janedoe.jpg",
      "jobTitle": "Professor",
      "name": "Jane Doe",
      "telephone": "(425) 123-4567",
      "url": "http://www.janedoe.com"
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-01-12T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.com/issuers/keys/1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TCYt5XsITJX1CxPCT8yAV-TVkIEq_Pb
  }
}
```

Figure 21. Sample Verifiable Credential: Business Card based on the Person schema example from Schema.org[6]

[5] Based on the following Microsoft Office (Microsoft Word) template: https://templates.office.com/en-ca/eighties-business-cards-tm34078835).

[6] Based on the sample data in JSON-LD version of Example 1 located at https://schema.org/Person#eg-0001

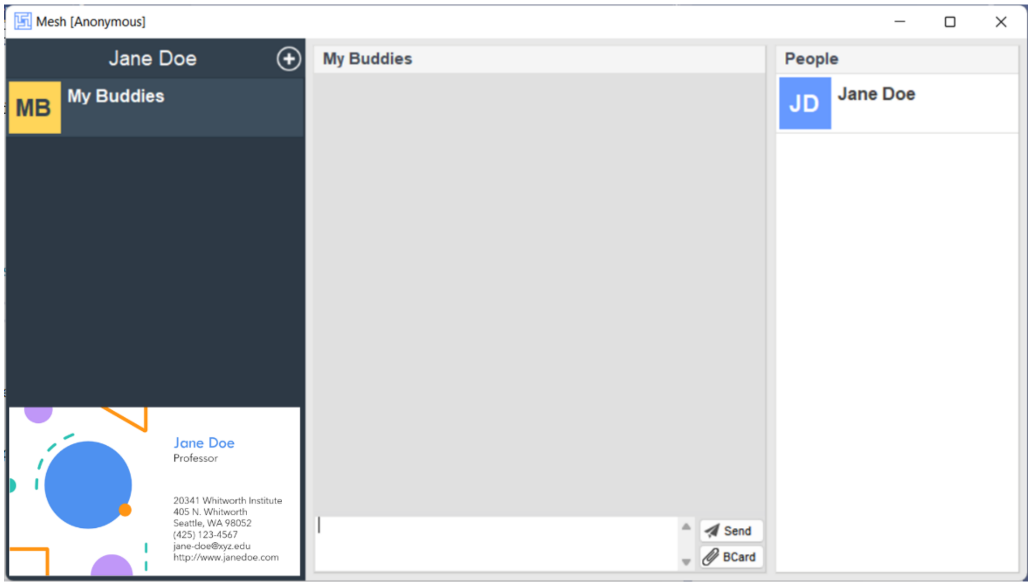
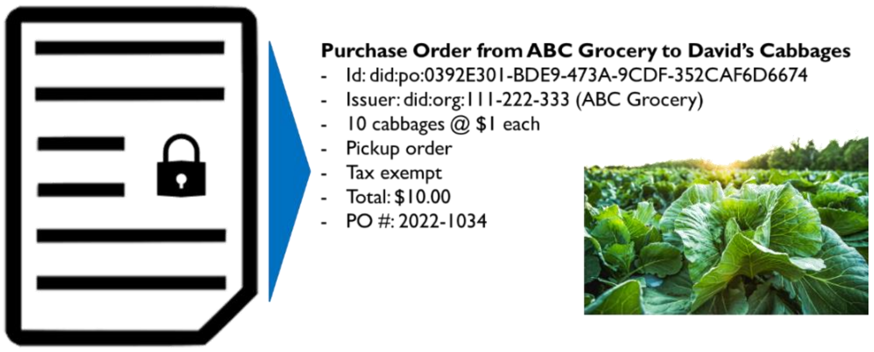


Figure 22. Level 3 DIDComm Agent Chat Sample App using Schema.org Person Verifiable Credential Attachments[7]

Purchase Order for 10 cabbages



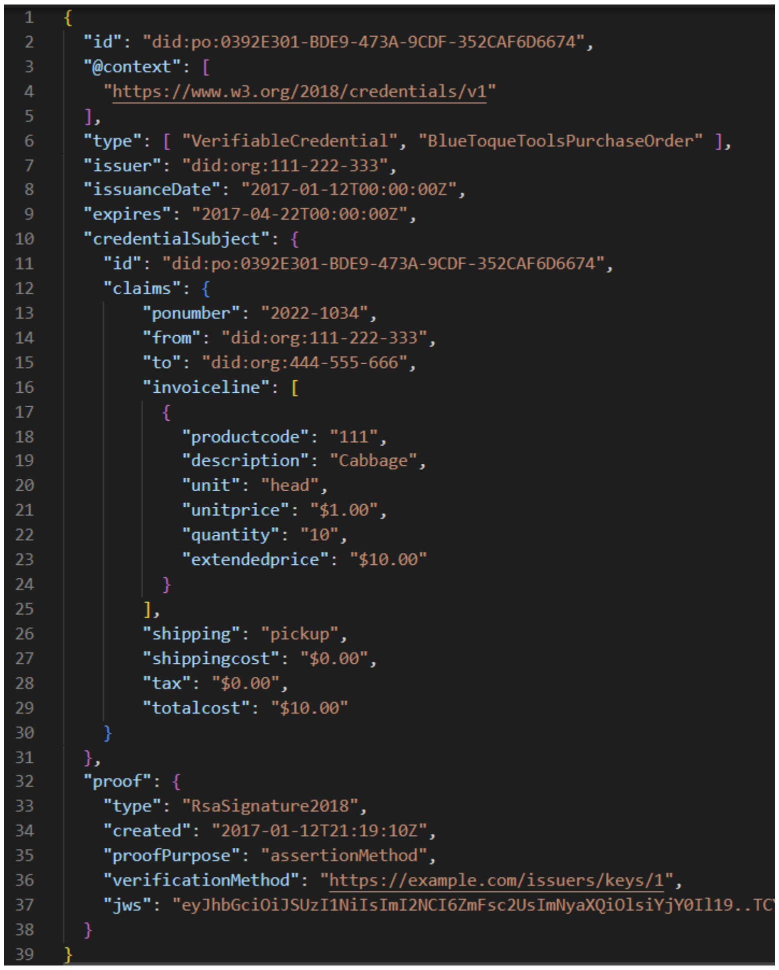
Purchase Order from ABC Grocery to David's Cabbages
- Id: did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674
- Issuer: did:org:111-222-333 (ABC Grocery)
- 10 cabbages @ $1 each
- Pickup order
- Tax exempt
- Total: $10.00
- PO #: 2022-1034

Figure 23. Rendering of a Purchase Order Verifiable Credential

Serialization of a Verifiable Credential using JSON

If you were able to investigate the JSON serialization of the above purchase order Verifiable Credential, it might look like the JSON in the following figure.

[7] The source code for the chat application used in this mock-up can be found here: https://github.com/TechnitiumSoftware/Mesh.



```
{
  "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [ "VerifiableCredential", "BlueToqueToolsPurchaseOrder" ],
  "issuer": "did:org:111-222-333",
  "issuanceDate": "2017-01-12T00:00:00Z",
  "expires": "2017-04-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
    "claims": {
        "ponumber": "2022-1034",
        "from": "did:org:111-222-333",
        "to": "did:org:444-555-666",
        "invoiceline": [
          {
            "productcode": "111",
            "description": "Cabbage",
            "unit": "head",
            "unitprice": "$1.00",
            "quantity": "10",
            "extendedprice": "$10.00"
          }
        ],
        "shipping": "pickup",
        "shippingcost": "$0.00",
        "tax": "$0.00",
        "totalcost": "$10.00"
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-01-12T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.com/issuers/keys/1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TC
  }
}
```

Figure 24. Sample Verifiable Credential: Purchase Order for 10 Cabbages

The Verifiable Credential was self-issued by ABC Grocery. The Verifiable Credential is a purchase order for 10 heads of cabbages, valued at $1 per head, tax-exempt, with an extended price and total cost of $10.00. The purchase order (Verifiable Credential) was from ABC Grocery (discernible by the “from” attribute) and directed to David’s Cabbages (whose decentralized identifier is the value of the “to” attribute).

The above scenario might look like the diagram below (produced using DIDComm Notation). A deep dive into this scenario can be found in the section Layer 3 DIDComm Agent with Verifiable Credential Attachments Model beginning on page 56.

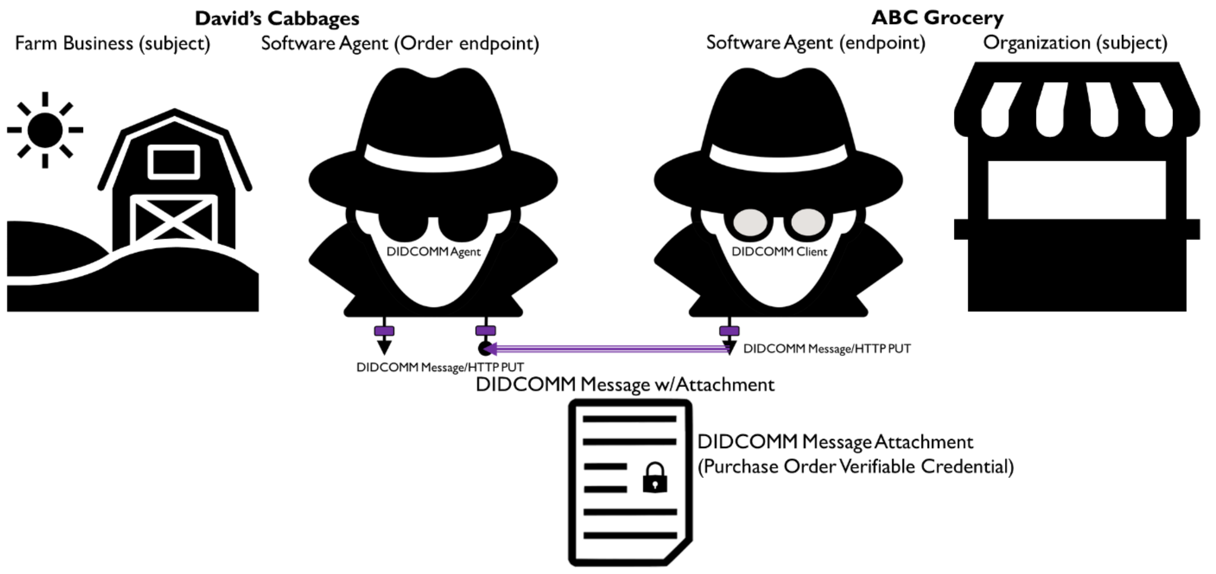

David's Cabbages
Farm Business (subject)
Software Agent (Order endpoint)
ABC Grocery
Software Agent (endpoint)
Organization (subject)
DIDCOMM Agent
DIDCOMM Client
DIDCOMM Message/HTTP PUT
DIDCOMM Message/HTTP PUT
DIDCOMM Message w/Attachment
DIDCOMM Message Attachment
(Purchase Order Verifiable Credential)

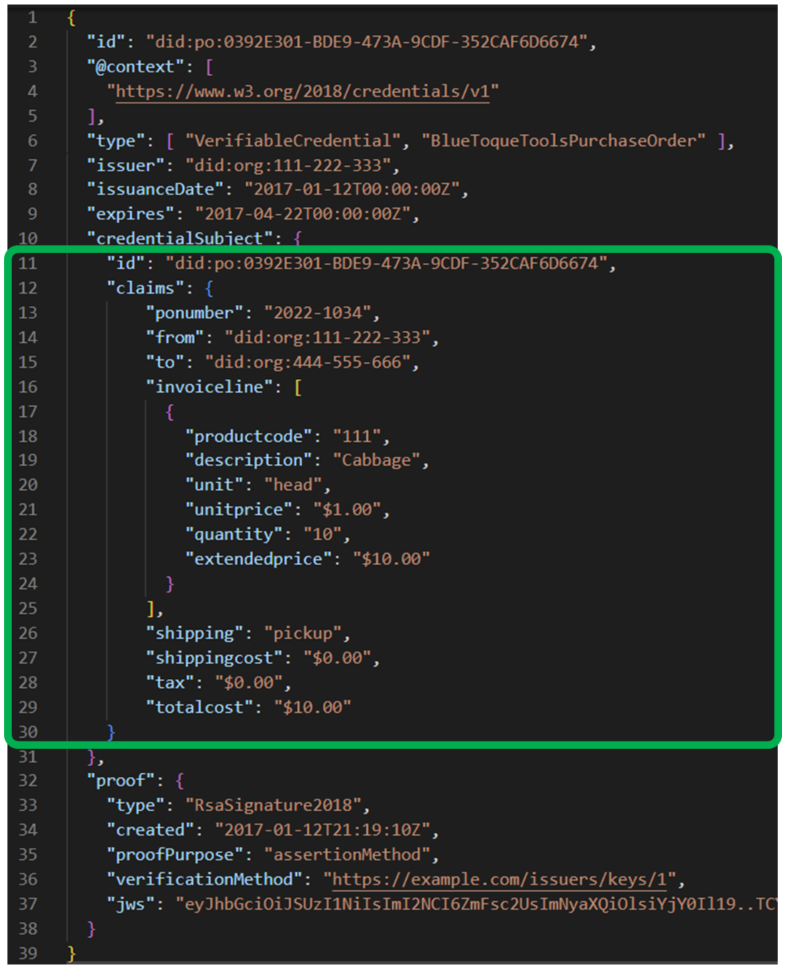
Figure 25. Layer 3 DIDComm Messaging with Verifiable Credential Attachments Model

Claims

In Figure 26. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Claims, there is a paragraph of JSON highlighted in green that starts with the attributes named "id" and "claims". This constitutes the application-specific business data that is being wrapped by the other sections of the Verifiable Credential JSON serialization.

This "id" attribute refers to, and by association, the concrete subject of this credential. In this case, the subject is the concrete version of the purchase order stored in a financial system somewhere at ABC Grocery; perhaps, a cloud-based accounting platform like Salesforce, Square, or Microsoft Dynamics.

Under the "claims" attribute are the individual claims or name-value pairs that collectively present the data found on the concrete purchase order. For example, the purchase order number is "2022-1034". There is a single invoice line item for the order of 10 cabbages. The "productcode" for a head of cabbage is "111".



```
{
  "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [ "VerifiableCredential", "BlueToqueToolsPurchaseOrder" ],
  "issuer": "did:org:111-222-333",
  "issuanceDate": "2017-01-12T00:00:00Z",
  "expires": "2017-04-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
    "claims": {
        "ponumber": "2022-1034",
        "from": "did:org:111-222-333",
        "to": "did:org:444-555-666",
        "invoiceline": [
          {
            "productcode": "111",
            "description": "Cabbage",
            "unit": "head",
            "unitprice": "$1.00",
            "quantity": "10",
            "extendedprice": "$10.00"
          }
        ],
        "shipping": "pickup",
        "shippingcost": "$0.00",
        "tax": "$0.00",
        "totalcost": "$10.00"
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-01-12T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.com/issuers/keys/1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TC
  }
}
```

Figure 26. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Claims

The remainder of the purchase order Verifiable Credential serves the following purposes:

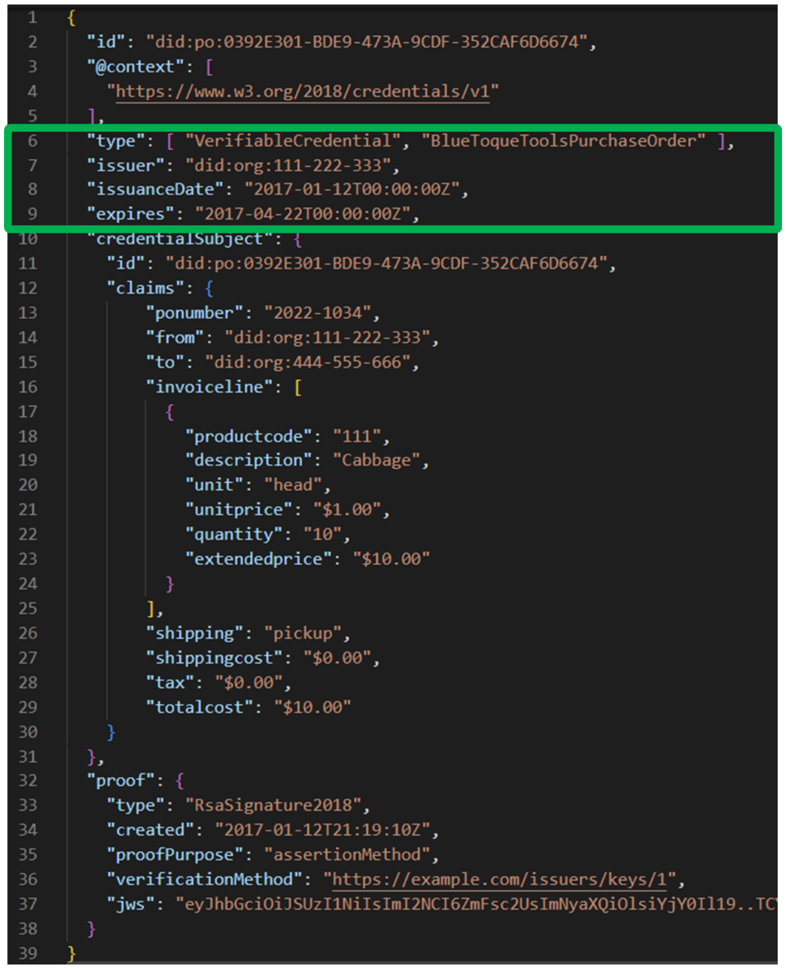
- Describes the specific Verifiable Credential type, who the issuing organization is, when it was issued, when the Verifiable Credential expires, etc.
- Provide a verifiable digital signature or “proof” for the entire Verifiable Credential (except, of course, for the “proof” itself)
- Encapsulate the Verifiable Credential with the equivalent of a digital envelope containing the purchase order data; sealed on the outside with the “proof”. This structure enables a Receiver Agent to determine that the Verifiable Credential (and hence the purchase order) is both authentic and that it hasn’t been tampered with.

Label (Metadata)

In Figure 27. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Label (Metadata), the paragraph of JSON highlighted in green is the Label containing metadata (descriptive information) for the Claims. This metadata includes:

- The Verifiable Credential “type” and subtype(s)
- The identifier of the “issuer”

- When it was issued and when it expires



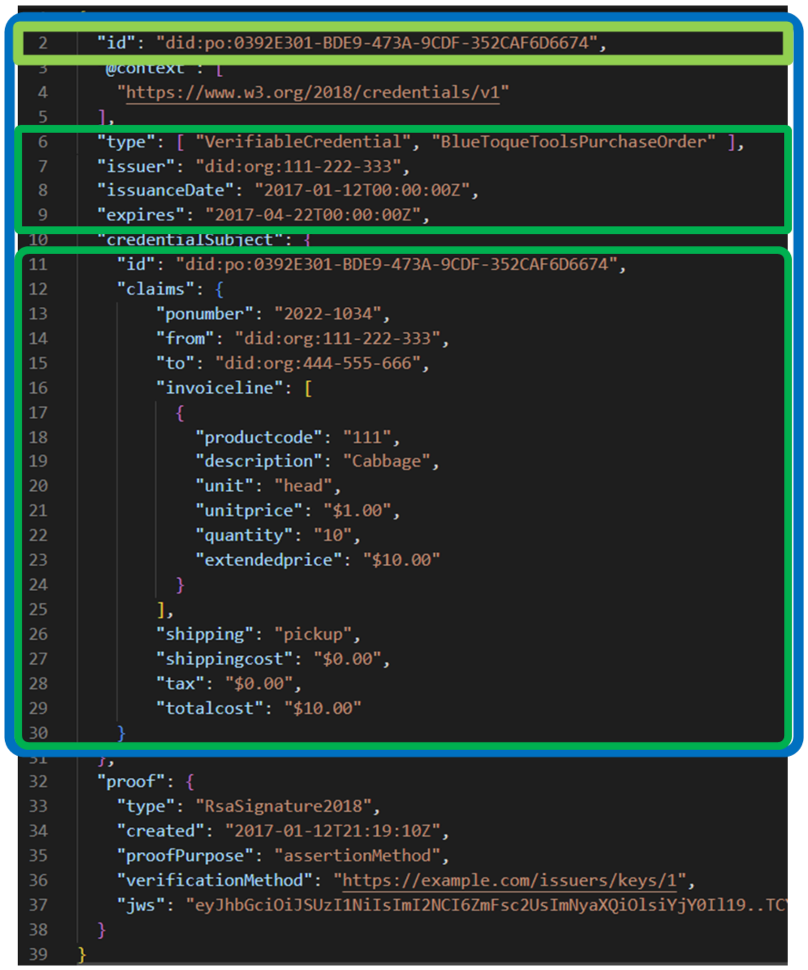
```
{
  "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [ "VerifiableCredential", "BlueToqueToolsPurchaseOrder" ],
  "issuer": "did:org:111-222-333",
  "issuanceDate": "2017-01-12T00:00:00Z",
  "expires": "2017-04-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
    "claims": {
        "ponumber": "2022-1034",
        "from": "did:org:111-222-333",
        "to": "did:org:444-555-666",
        "invoiceline": [
          {
            "productcode": "111",
            "description": "Cabbage",
            "unit": "head",
            "unitprice": "$1.00",
            "quantity": "10",
            "extendedprice": "$10.00"
          }
        ],
        "shipping": "pickup",
        "shippingcost": "$0.00",
        "tax": "$0.00",
        "totalcost": "$10.00"
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-01-12T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.com/issuers/keys/1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TC
  }
}
```

Figure 27. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Label (Metadata)

Envelope

The Envelope is an abstraction that includes all of the parts described above (less the proof). The Envelope is the portion of the Verifiable Credential that is used to produce a content hash and subsequent digital signature using the private key of the issuer. In Figure 28. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Envelope, the Envelope is highlighted in blue.

The Envelope also includes an outer “id” attribute. Think of this as an identifier that appears on the outside of the Envelope. Often, but not always, the value of this outer “id” attribute is the same as the value of the “id” attribute that appears in the Claims.

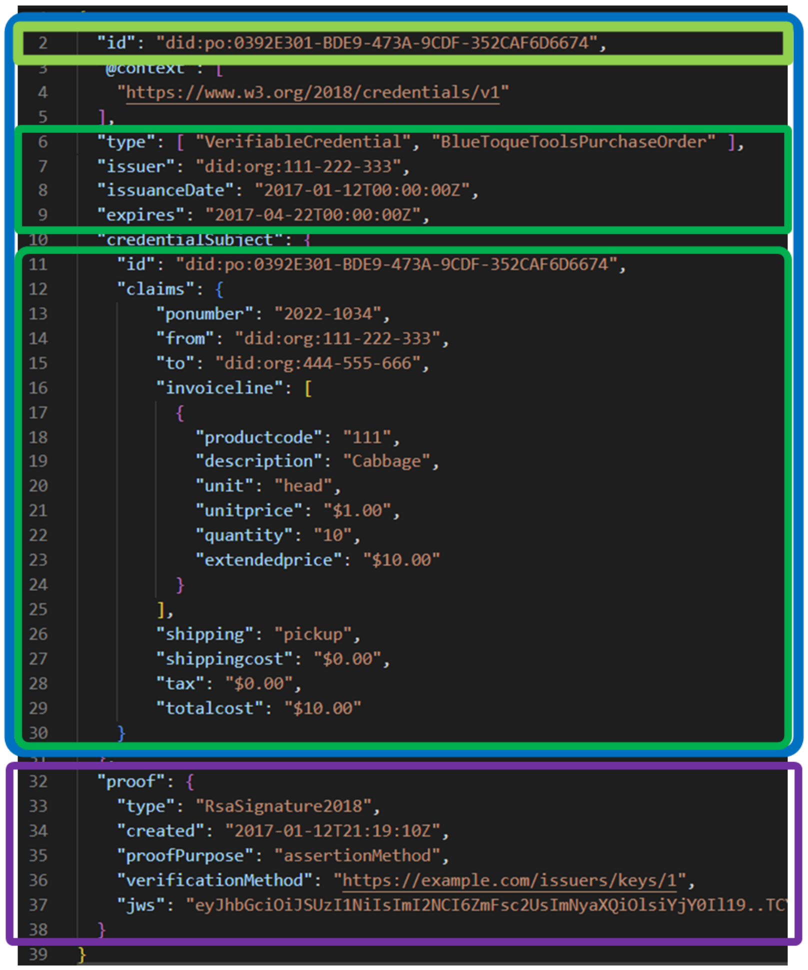


```
{
  "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [ "VerifiableCredential", "BlueToqueToolsPurchaseOrder" ],
  "issuer": "did:org:111-222-333",
  "issuanceDate": "2017-01-12T00:00:00Z",
  "expires": "2017-04-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
    "claims": {
        "ponumber": "2022-1034",
        "from": "did:org:111-222-333",
        "to": "did:org:444-555-666",
        "invoiceline": [
          {
            "productcode": "111",
            "description": "Cabbage",
            "unit": "head",
            "unitprice": "$1.00",
            "quantity": "10",
            "extendedprice": "$10.00"
          }
        ],
        "shipping": "pickup",
        "shippingcost": "$0.00",
        "tax": "$0.00",
        "totalcost": "$10.00"
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-01-12T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.com/issuers/keys/1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TC
  }
}
```

Figure 28. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Envelope

Seal (Proof)

The final component of a Verifiable Credential is the “proof”. In Figure 29. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Proof, the Proof is highlighted in purple. As mentioned earlier, the “proof” is computed based first on the content hash of the contents of the Envelope; subsequently, a digital signature of the hash is computed using the private key of the issuer. This digital signature plus additional metadata in the “proof” property is needed to support verification of the “proof” (and by implication, the Verifiable Credential and the purchase order data).



```
  "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [ "VerifiableCredential", "BlueToqueToolsPurchaseOrder" ],
  "issuer": "did:org:111-222-333",
  "issuanceDate": "2017-01-12T00:00:00Z",
  "expires": "2017-04-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:po:0392E301-BDE9-473A-9CDF-352CAF6D6674",
    "claims": {
        "ponumber": "2022-1034",
        "from": "did:org:111-222-333",
        "to": "did:org:444-555-666",
        "invoiceline": [
          {
            "productcode": "111",
            "description": "Cabbage",
            "unit": "head",
            "unitprice": "$1.00",
            "quantity": "10",
            "extendedprice": "$10.00"
          }
        ],
        "shipping": "pickup",
        "shippingcost": "$0.00",
        "tax": "$0.00",
        "totalcost": "$10.00"
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-01-12T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.com/issuers/keys/1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TC
  }
}
```

Figure 29. Sample Verifiable Credential (Purchase Order for 10 Cabbages): Proof

DIDCOMM AGENT ARCHITECTURE REFERENCE MODEL (DIDCOMM-ARM): OVERVIEW

The DIDComm Agent Architecture Reference Model (DIDComm-ARM) is a layered architecture reference model designed to aid software architects and developers by making it easier for them to understand how to design and create DIDComm Agent-based software systems. Visually, the DIDComm-ARM is described (modeled) using DIDComm Notation. The elements of DIDComm Notation were described in the first half of this whitepaper. The categories of elements found in DIDComm Notation include:

- REST/HTTP Origin and Endpoint Interfaces (Layer 0)
- REST/HTTP Clients, Agents, and Services (Layer 0)
- Decentralized Identifier Registries (DID Registries) (Layer 1)
- DIDComm Origin and Endpoint Interfaces (Layer 2)
- DIDComm Clients and Agents (Layer 2)
- Verifiable Credentials and Verifiable Credential Attachments (Layer 3)

These elements are used to define and describe the layers of the DIDComm-ARM found in the remainder of this document.

Layered DIDComm-ARM Model

The DIDComm-ARM contains the following layered architecture models:

- Layer 0 REST/HTTP Agent Model
- Layer 1 DID Addressable REST/HTTP Agent Model
- Layer 2 DIDComm Agent Model
- Layer 3 DIDComm Agent with Verifiable Credential Attachments Model
- Layer 4 DIDComm Agent Mesh Network Model

The DIDComm-ARM layers can be organized into a feature matrix based on the characteristics that define each layer based on the dimensions: DID Addressability and DIDComm Messaging (as illustrated below).

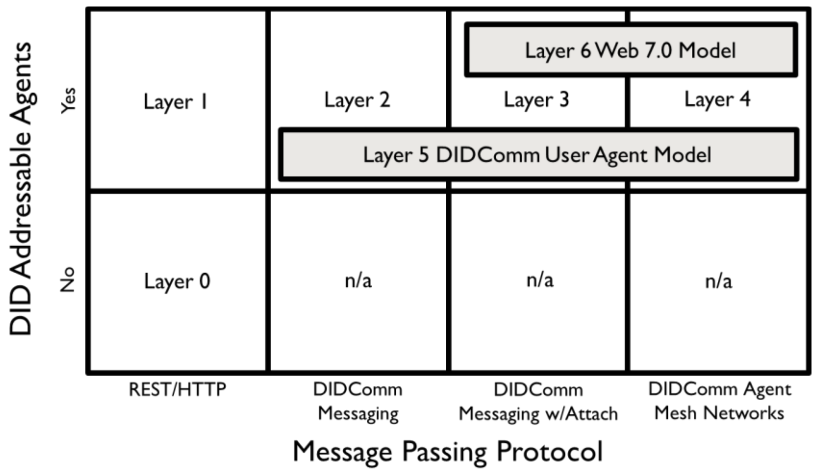


Figure 30. DIDComm-ARM Feature Matrix

NOTE: Two late additions to the current version of the DIDComm-ARM include:

- Layer 5 DIDComm User Agent Model
- Layer 6 Web 7.0 DIDComm Agent Architecture Model

NOTE: The descriptions of Layer 5 and Layer 6 are not included in the main body of this whitepaper. A description of Layer 5 has been added as APPENDIX A – Layer 5 DIDComm User Agent Model on page 64. A description of Layer 6 has been added as APPENDIX B – Layer 6 on page 69.

DIDComm-ARM Scenarios

Each of the DIDComm-ARM layers is described in terms of a characteristic scenario:

- Layer 0 REST/HTTP Agent Model Overview
- Layer 1 DID Addressable REST/HTTP Agent Model Overview
- Level 2 DIDComm Agent Model Overview
- Layer 3 DIDComm Agent with Verifiable Credential Attachments Model Overview
- Layer 4 DIDComm Agent Mesh Network Model Overview

A brief overview of each of these scenarios is provided in the following sections.

Layer 0 REST/HTTP Agent Model Overview

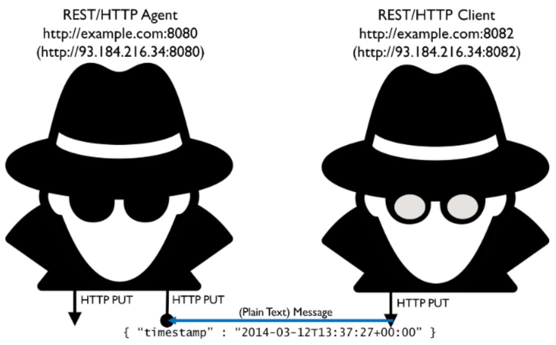


Figure 31. Layer 0 REST/HTTP Agent Model

Layer 0 REST/HTTP Agent model is the simplest scenario: a plain text message is transmitted "in the clear" between a REST/HTTP Client and REST/HTTP Agent. The REST/HTTP Client needs to know or determine the Internet address of the REST/HTTP Agent Receiver Agent. In this scenario, the Internet address is determined using conventional means (e.g. hardcoded in the application or an application settings file, querying a localhosts file or a local or remote DNS server). A DID Registry is not used nor is it needed in this scenario.

The purpose of Layer 0 is to provide a common technical foundation that a vast majority of software architects and developers around the world will be familiar with - regardless of their preferred technology stack. Each subsequent layer then builds off this common technical foundation and the other layers in the DIDComm-ARM technology stack.

Layer 1 DID Addressable REST/HTTP Agent Model Overview

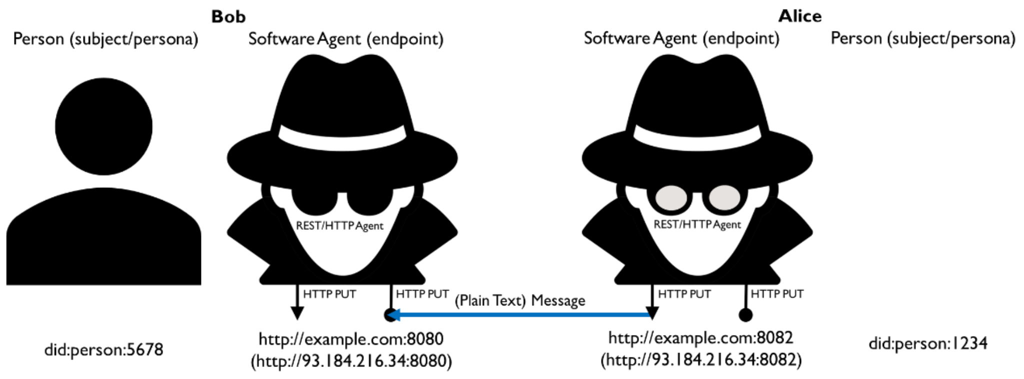


Figure 32. Layer 1 DID Addressable REST/HTTP Agent Model: Personification

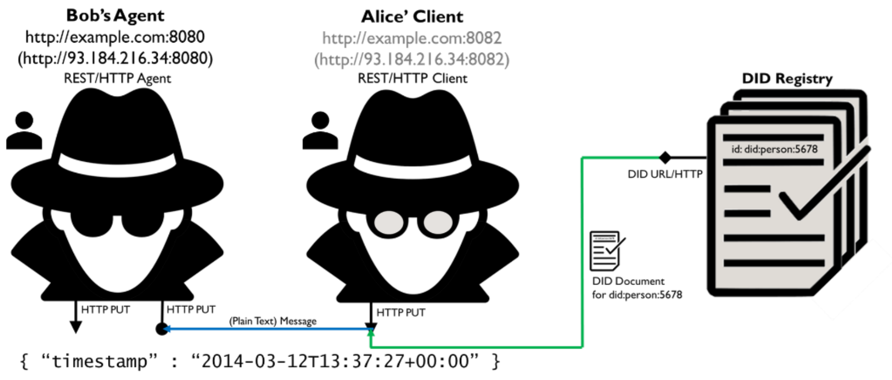


Figure 33. Layer 1 DID Addressable REST/HTTP Agent Model

The Layer 1 DID Addressable REST/HTTP Agent model builds on top of Layer 0 by including a DID Registry and associating, in this scenario, a person to each of the agents and using the person's decentralized identifier (DID) to lookup the Internet address of the Receiver REST/HTTP Agent, in this case, Bob.

The purpose of Layer 1 is to introduce a single decentralized identifier-based concept without having to engage with the other components of the DID Communications (DIDComm) framework. That is, how decentralized identifiers (DIDs) can be used to discover, locate, and interact with a REST/HTTP Agent service endpoint (and in subsequent layers, DIDComm Agent service endpoints).

Layer 2 DIDComm Agent Model Overview

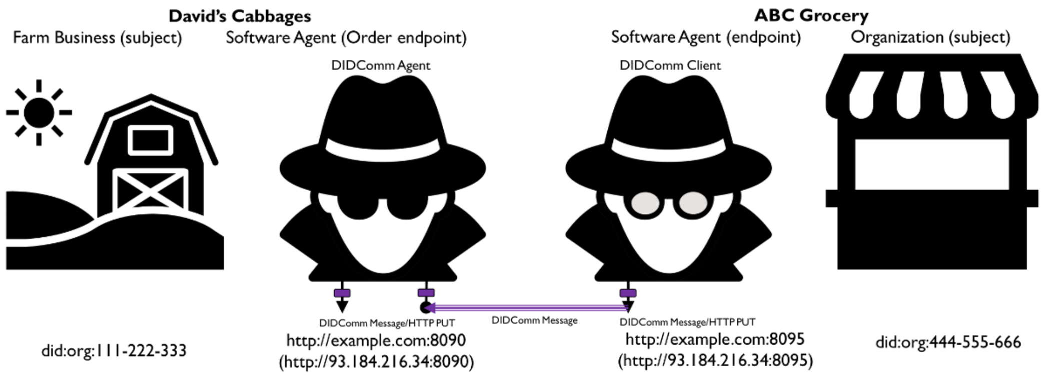


Figure 34. Level 2 DIDComm Agent Model: Personification

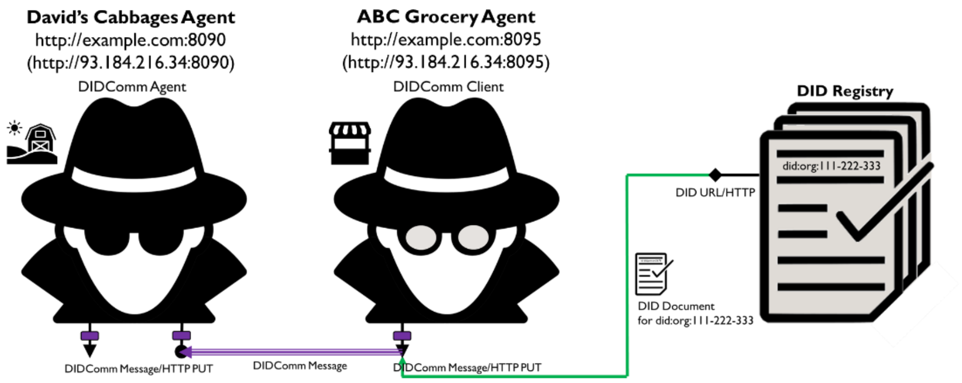


Figure 35. Level 2 DIDComm Agent Model

The Layer 2 DIDComm Agent model builds on top of the Layer 1 DID Addressable REST/HTTP Agent scenario by replacing the REST/HTTP plain text messages (and protocol) with authenticated, encrypted DIDComm messages. The Sender Agent in this scenario is ABC Grocery's DIDComm Client; the Receiver Agent is David's Cabbages' DIDComm Agent.

The purpose of Layer 2 is to introduce the single concept of DIDComm Messaging (and DIDComm Messages). Layer 2 differs from Layer 1 in that DIDComm Messaging supersedes the use of plain old REST/HTTP simple text-based messages.

Layer 3 DIDComm Agent with Verifiable Credential Attachments Model Overview

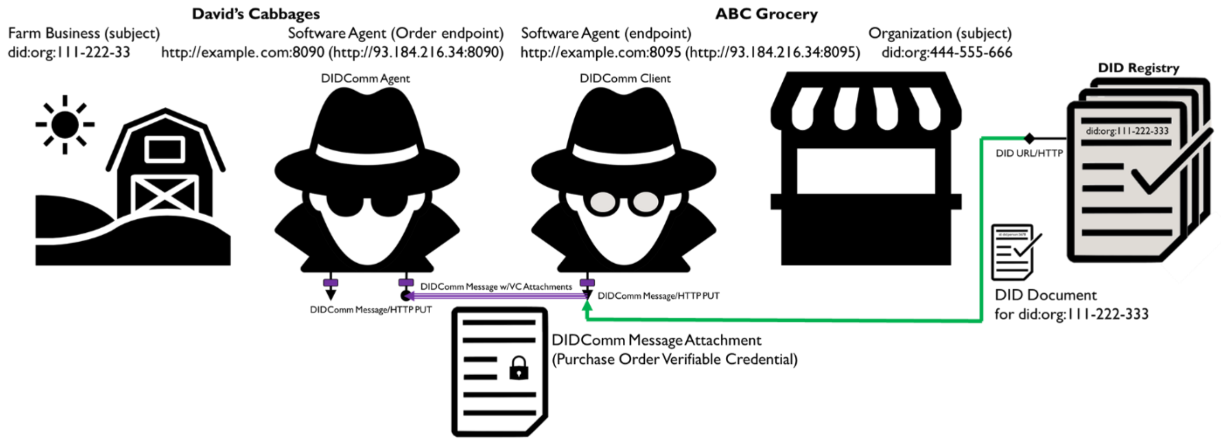


Figure 36. Layer 3 DIDComm Agent with Verifiable Credential Attachments Model

The Layer 3 DIDComm Agent with Verifiable Credential Attachments Model builds on top of the Layer 2 DIDComm Agent scenario by enabling the use of DIDComm Message Attachments and using this capability to attach a Verifiable Credential to a DIDComm Message. In this specific scenario, a purchase order Verifiable Credential is issued by ABC Grocery's DIDComm Client and sent as a DIDComm Message Attachment to David's Cabbages' DIDComm Agent.

The purpose of Layer 3 is to introduce the single additional capability of DIDComm Message Attachments (and to use Verifiable Credentials as a key (critical) example of DIDComm Message Attachment).

Layer 4 DIDComm Agent Mesh Network Model Overview

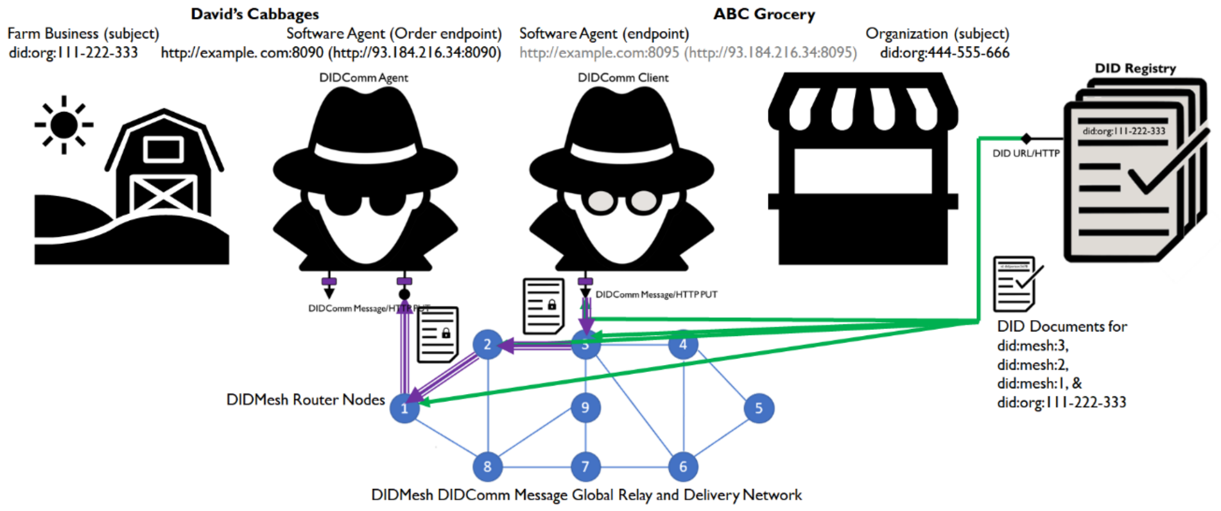


Figure 37. Layer 4 DIDComm Agent Mesh Network

The Layer 4 DIDComm Agent Mesh Network Model builds on the previous layers of the DIDComm-ARM by replacing the point-to-point DIDComm Message routing between a client and an agent or an agent and a second agent with a network-based message routing solution to support mesh network routing of DIDComm Messages. This model can be thought of as an extension or elaboration of how Mediators and Relays work with the DIDComm Routing pattern (https://didcomm.org/book/v2/routing).

The purpose of Layer 4 is to introduce the concept of DIDComm Message Routing (and in particular, use these concepts to describe the DIDComm Network, a DIDComm Message-based global relay and delivery network model).

LAYER 0 REST/HTTP AGENT MODEL

The Layer 0 REST/HTTP Agent Model may seem trivial and unnecessary, but it provides a foundational basis and structure for the subsequent layers of the DIDComm-ARM.

The basic scenario consists of a REST/HTTP Client wanting to send a plain text message to a REST/HTTP Agent as depicted below. A simple coded example can be found here: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-7.0#examples.

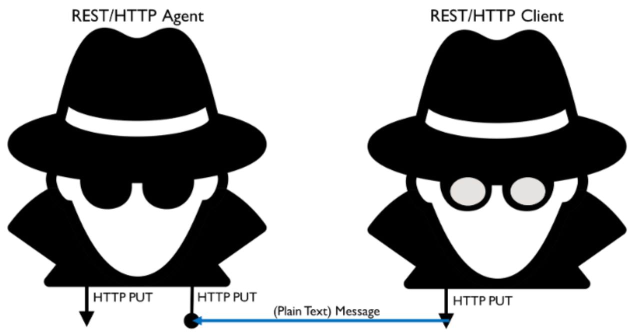


Figure 38. Layer 0 REST/HTTP Agent Model

The specific configuration for this scenario is depicted in this second figure.

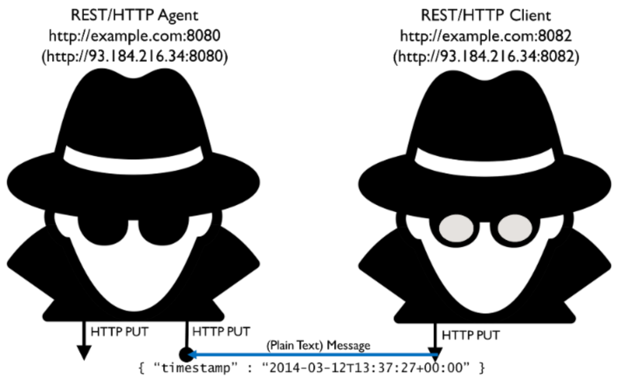


Figure 39. Layer 0 REST/HTTP Agent Model

To be able to send a message from the Sender Agent REST/HTTP Client to the Receiver REST/HTTP Agent, the instance of the HTTPClient software needs the IP address and port of the Receiver Agent. There are several ways this can be accomplished:

- Hardcode the (numerical) IP address and port number in the Sender Client application (or one of its configuration files) and encode the IP address and port number in a URL passed to HTTPClient.
- Hardcode a domain name that resolves to the IP address in the localhosts file associated with the network stack and encode the domain name (and port number) in a URL passed to the HTTPClient.
- Register the domain name and IP address in a locally deployed DNS server.
- Register the domain name and IP address in an Internet-hosted DNS server (e.g. Network Solutions, GoDaddy, etc.).

NOTE: The key message is that the address of the REST/HTTP Agent's service endpoint needs to ultimately be resolved to a numerical IP address and port number. The IP address corresponds to a physical or virtual network interface on the computer system hosting the Agent's executing code. The port number corresponds to either a well-known port number (https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml) or a locally chosen port number (e.g. 8080, 8082).

> *Port numbers are assigned in various ways, based on three ranges: System Ports (0-1023), User Ports (1024-49151), and the Dynamic and/or Private Ports (49152-65535); the different uses of these ranges are described in [RFC6335]. According to Section 8.1.2 of [RFC6335], System Ports are assigned by the "IETF Review" or "IESG Approval" procedures described in [RFC8126]. User Ports are assigned by IANA using the "IETF Review" process, the "IESG Approval" process, or the "Expert Review" process, as per [RFC6335]. Dynamic Ports are not assigned. (https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml)*

This is a key element of the Layer 1 DID Addressable REST/HTTP Agent Model (described in the next section) where instead of the Sender Agent having an IP address (and port number) available or a domain name (and port), the Sender Agent only knows the decentralized identifier (DID) associated with the concrete entity associated with the specific Receiver REST/HTTP Agent.

Layer 0 REST/HTTP Agent Model Workflow

The following is an outline of the workflow depicted in the above model.

Goal: Send a plain text message from a REST/HTTP Client to a REST/HTTP Agent knowing only the Agent's service endpoint Internet address and port number (e.g. http://example.com/8080).

1. Resolve the Receiver Agent's service endpoint Internet address and port number to an IP address and port number.
2. Prepare the plain text of the message to be sent (e.g. plain old character string, JSON serialization of a plain old class object (POCO), etc.).
3. Instantiate an instance of an HTTPClient
4. Call the Post method on the HTTPClient instance to send the plain text message to the resolved IP address and port number to the Receiver Agent as an HTTP request.
5. Wait for the (empty) response.
6. Check the HTTP error code returned.
7. Receiver Agent: Receive, accept, and process the plain text message received from the Sender Agent

LAYER 1 DID ADDRESSABLE REST/HTTP AGENT MODEL

In the Layer 1 DID Addressable REST/HTTP Agent Model, the Sender Agent has no a priori knowledge of the Internet address and port number of the Receiver Agent's REST/HTTP Agent. The only thing the Sender Client or Agent knows is the decentralized identifier of the Subject whose REST/HTTP Agent is the intended Receiver Agent.

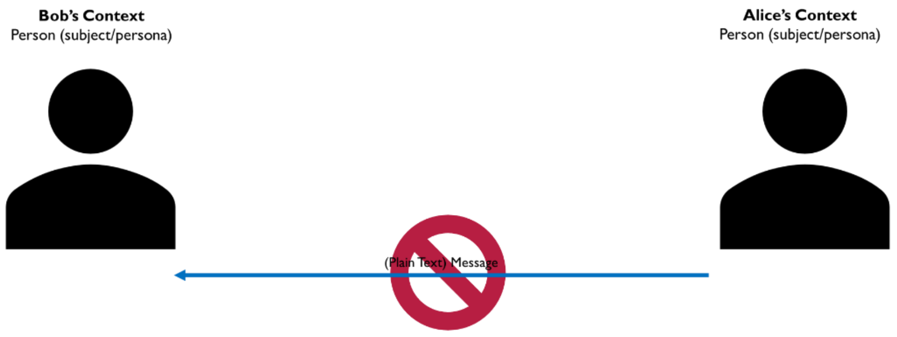


Figure 40. Each Personifications needs to be paired with a DIDComm Agent to support Message Passing

What do Decentralized Identifiers Identify?

A decentralized identifier should be interpreted as a unique reference or identifier to a concrete entity (aka subject): a person, an organization, a business document (purchase order, invoice, etc.), an education credential, a car, a boat, a house, a software module, a deployed instance of a software module, etc.

Everything else in a decentralized identifier-based software system is addressed by dereferencing or resolving the decentralized identifier to obtain a reference or address of the other types of system entities (e.g. a DID Document, Revocation List entry, Service Endpoint addresses (via a second level of indirection through the DID Document)).

For example, the (outer) "id" field in a Verifiable Credential should be the unique identifier for the concrete Verifiable Credential entity. The (inner) "id" associated with the "credentialSubject" is the identifier for the "inner credential", "business credential" or "payload" embedded inside the Verifiable Credential.

The following are examples of subjects and their software agents that are designed to emphasize the above points.

People and Agents

Alice and Bob are people. In addition, each of Alice and Bob have their own software agent. Each software agent has one or more Endpoint interfaces that are used to receive messages from other parties. Usually, an Agent will also have an Origin interface that is capable of sending messages to another agent's Endpoint interface (as illustrated below).

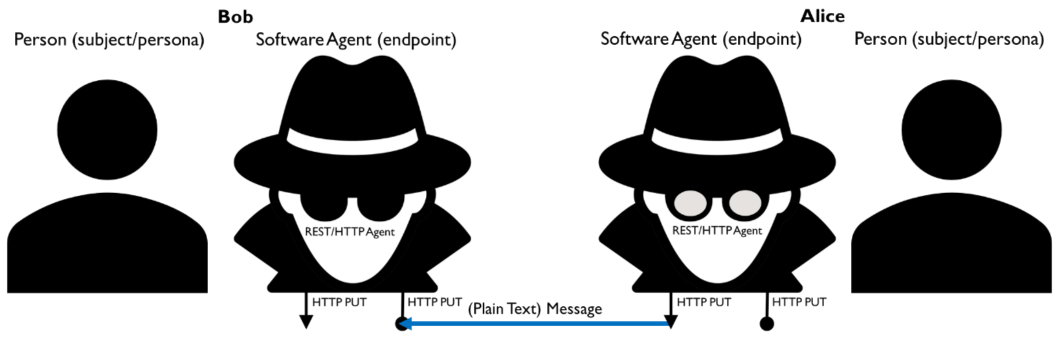


Figure 41. People and Agents

In a decentralized identifier-based software system, each person is expected to have one or more unique decentralized identifiers. In the example below, Alice, the person, has the decentralized identifier did:person:1234 and Bob, also a person, has a decentralized identifier of did:person:5678.

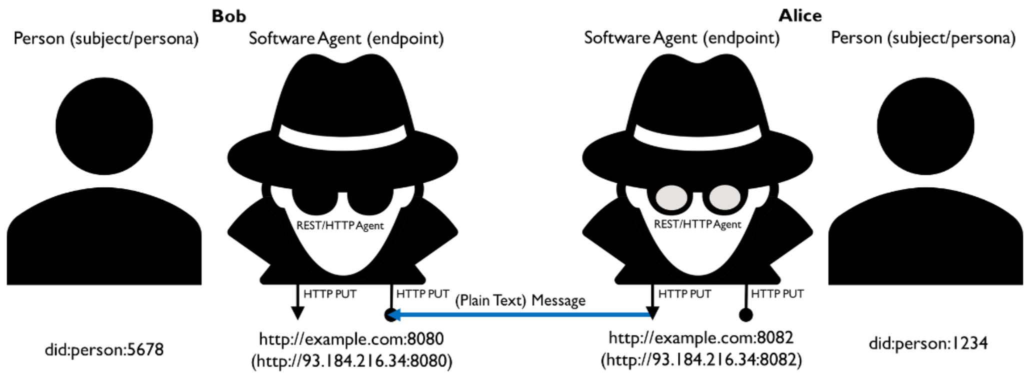


Figure 42. People, Agents, and Decentralized Identifiers (DIDs)

It is possible for an agent and/or an agent's Endpoint interface to also have assigned decentralized identifiers but this is usually not the case except in special, atypical application scenarios (e.g. software agent versioning, software development, etc.).

Institutions and People

Similar to the scenario above, an institution like the hypothetical Thrift Bank can have a decentralized identifier (e.g. did:org:123-456-789). This DID is the unique identifier for the concrete (but hypothetical) banking institution. In reality, a banking institution might have 100s, 10,000s, and quite possibly millions of decentralized identifiers within its scope that are associated with bank subsidiaries, divisions and departments, employees, banking and investment accounts, financial products, banking branches, ATMs, etc.

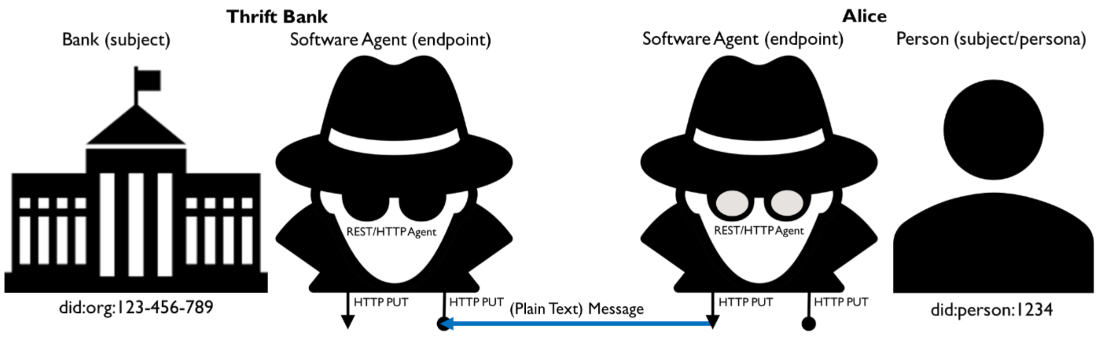


Figure 43. Institutions, People, and Agents

Small Businesses and People

Companies – all businesses small, medium, and large – will similarly have 100s, 10,000s, and quite possibly millions of decentralized identifiers within their scope. In the example below, David's Cabbages, an agri-products company, has the decentralized identifier did:org:111-222-333. Again, this is the DID associated with David's Cabbages, a concrete business entity.

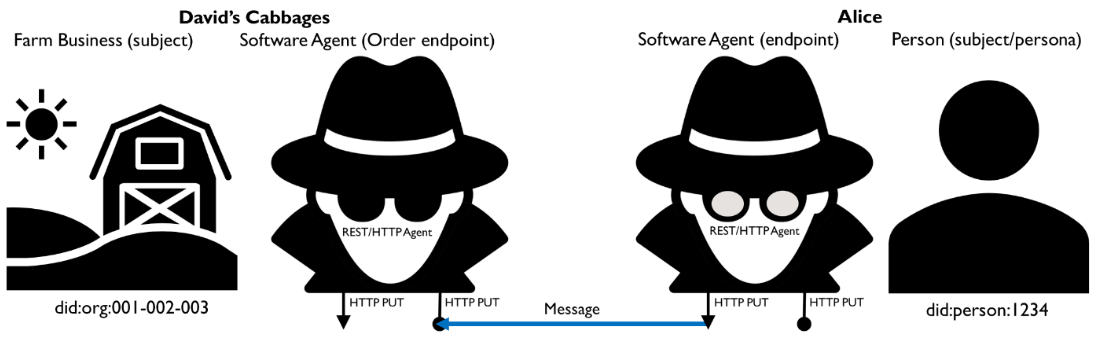


Figure 44. Small Businesses, People, and Agents

Resolving a Service Endpoint URL from a Decentralized Identifier (DID)

How can a Sender Client determine the Internet address and port number of the Receiver Agent given only the DID for the Subject associated with the Receiver Agent? The DIDComm Notation model for this scenario is depicted in the following figure.

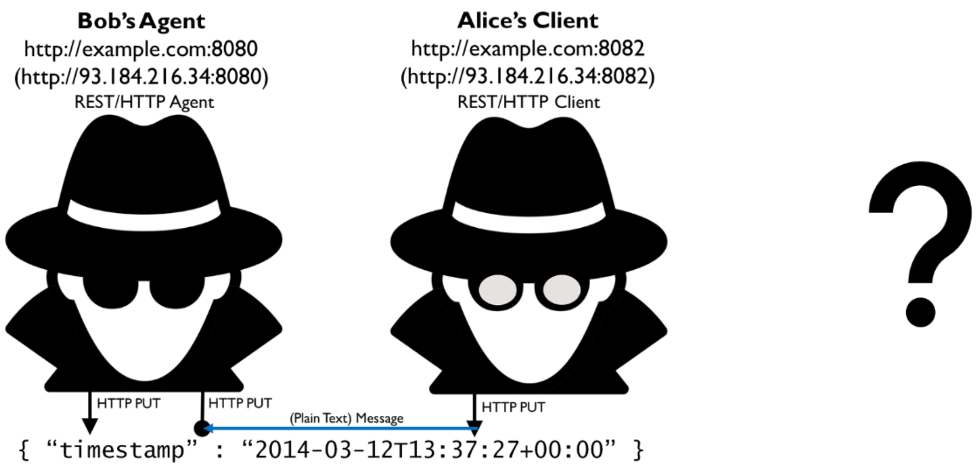


Figure 45. Layer 1 DID Addressable REST/HTTP Agent Configuration

Based on the above configuration of the model, more specific details about the Layer 1 DID Addressable REST/HTTP Agent Model scenario include:

1. Alice is a Person who has an association with the following DID: did:person:1234. Alice is the subject of this DID.
2. Bob is a Person who has an association with a different DID: did:person:5678. Bob is the subject of this DID.
3. Alice's REST/HTTP Client, in the role of a Sender Agent, wants to send a plain text message whose value is the current time-of-day (a DateTime value) to Bob's REST/HTTP Agent.
4. The Internet address and port number for Alice's REST/HTTP Client is http://example.com:8082; which, in turn, further resolves to http://93.184.216.34:8082.
5. The Internet address and port number for Bob's REST/HTTP Agent is http://example.com:8085; which, in turn, further resolves to http://93.184.216.34:8085.

For Alice's REST/HTTP Client to determine the Internet address and port number of Bob's REST/HTTP Agent, Alice can rely on the services of a DID Registry (as illustrated in the following series of DIDComm Notation models). This assumes that Bob's decentralized identifier is registered in the DID Registry.

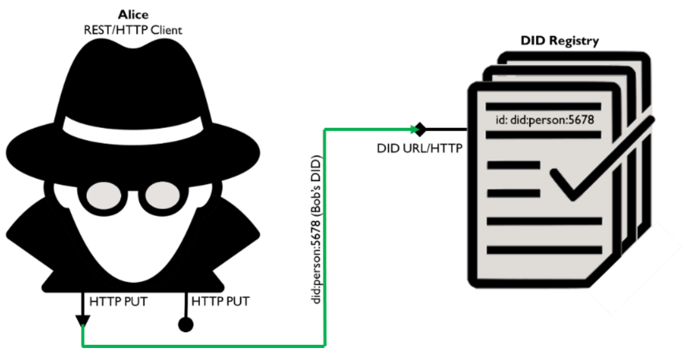


Figure 46. DID Resolution Request: Alice Requesting Bob's DID Document (did:person:5678)

As illustrated above, for Alice's REST/HTTP Client to resolve the Internet address and port number of Bob's REST/HTTP Agent, Alice's REST/HTTP Client can send a "resolve DID" request passing Bob's decentralized identifier (did:person:5678) as a parameter to the DID Registry's DID URL/HTTP Request/Response Interface.

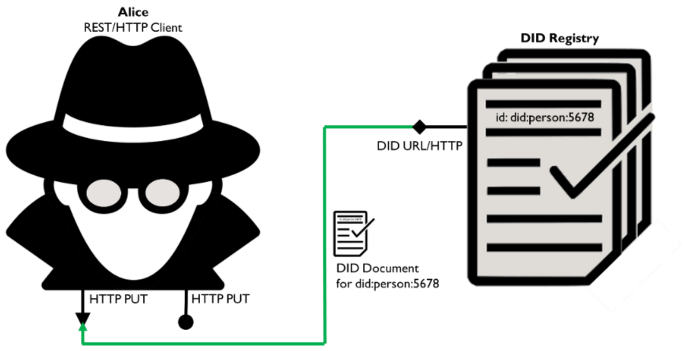
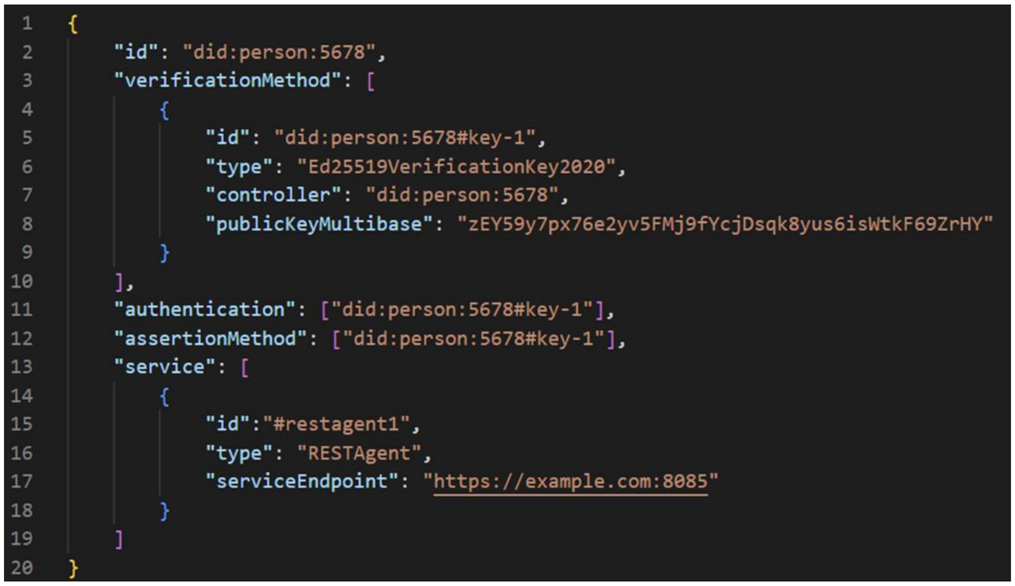


Figure 47. DID Resolution Response: Alice Receiving Bob's DID Document (did:person:5678)

The DID Registry will process the "resolve DID" request for DID did:person:5678 and return a JSON serialized response specific to did:person:5678 (again assuming Bob's DID is registered with the DID Registry). The JSON serialized response is called a DID Document.

DID Documents

An example of a prototypical DID Document that might be returned by the DID Registry in response to a "resolve DID" request for DID did:person:5678 appears in the following figure.



```
{
    "id": "did:person:5678",
    "verificationMethod": [
        {
            "id": "did:person:5678#key-1",
            "type": "Ed25519VerificationKey2020",
            "controller": "did:person:5678",
            "publicKeyMultibase": "zEY59y7px76e2yv5FMj9fYcjDsqk8yus6isWtkF69ZrHY"
        }
    ],
    "authentication": ["did:person:5678#key-1"],
    "assertionMethod": ["did:person:5678#key-1"],
    "service": [
        {
            "id":"#restagent1",
            "type": "RESTAgent",
            "serviceEndpoint": "https://example.com:8085"
        }
    ]
}
```

Figure 48. DID Document Response for did:person:5678 (example)

In the above DID Document returned as a response to the "resolve DID" request for did:person:5678, note the DID Document returns 4 types of information associated with Bob, the subject of did:person:5678:

i. the verification method (aka "public key" in most contexts),
ii. the authentication attribute,

iii. the assertionMethod attribute, and
iv. the most important attribute in the context of the Layer 1 DID Addressable REST/HTTP Agent scenario: the "serviceEndpoint" value (a URL).

It is the value of the 4th attribute "serviceEndpoint" that enables Alice's REST/HTTP Client to resolve the Internet address (and port number) for Bob's REST/HTTP Agent. We can now replace the question mark (?) in the original Layer 1 model: it is the DID Registry where Bob's DID is registered. The DID Registry has been added to the Layer 1 model in the following figure.

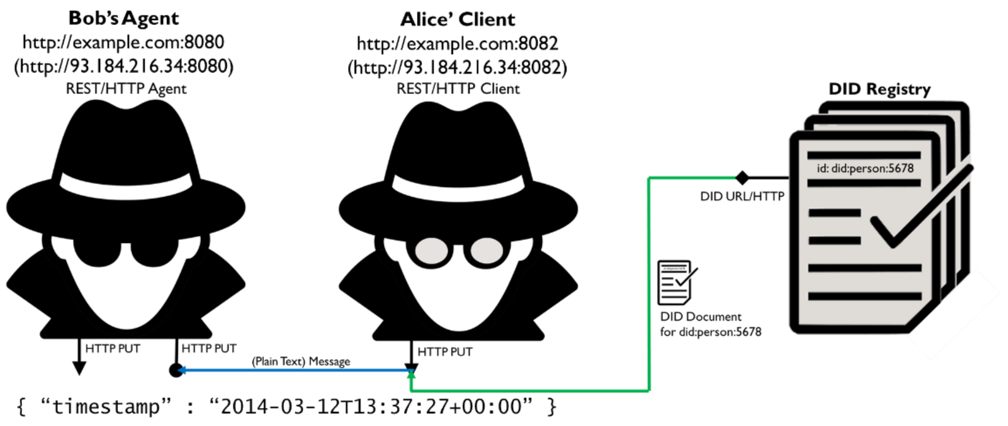


Figure 49. Layer 1 DID Addressable REST/HTTP Agent Model

Layer 1 DID Addressable REST/HTTP Agent Model Workflow

The following is an outline of the workflow depicted in the above model. The steps that are new (or changed) in this workflow relative to the Layer 0 REST/HTTP Agent Model are highlighted in bold.

1. **Verify that Bob's decentralized identifier did:person:5678 is registered on a particular DID Registry.**
2. **Resolve Bob's decentralized identifier did:person:5678 from the DID Registry to receive a copy of the DID Document associated with this DID.**
3. **Dereference the "serviceEndpoint" subelement of the appropriate "service" element in the received DID Document**
4. **Use the value of the "serviceEndpoint" subelement as the Agent's service endpoint Internet address (and port number if specified). If no port number is specified, assume it has a value of 80.**
5. Resolve the Receiver Agent's service endpoint Internet address and port number to an IP address and port number.
6. Prepare the plain text of the message to be sent (e.g. plain old character string, JSON serialization of a plain old class object (POCO), etc.).
7. Instantiate an instance of an HTTPClient
8. Call the Post method on the HTTPClient instance to send the plain text message to the resolved IP address and port number of the Receiver Agent as an HTTP request.
9. Wait for the (empty) response.
10. Check the HTTP error code returned.

11. Receiver Agent: Receive, accept, and process the plain text message received from the Sender Agent

LAYER 2 DIDCOMM AGENT MODEL

The Layer 2 DIDComm Agent Model builds on the Layer 1 DID Addressable REST/HTTP Agent Model by replacing the simple request-response REST/HTTP messaging paradigm with DIDComm Messaging. DIDComm Messaging enables message payloads to be authenticated and encrypted for secure, confidential, and private communications between a DIDComm Client and a DIDComm Agent or any pair of DIDComm Agents.

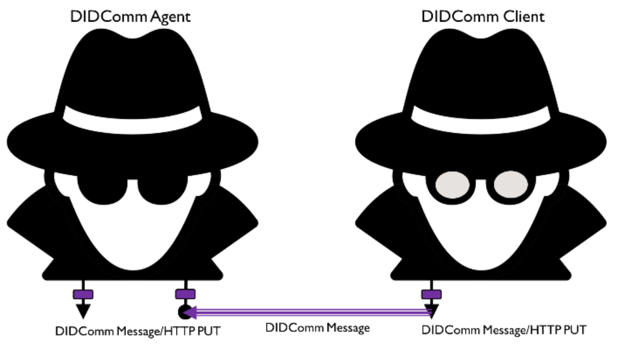


Figure 50. Layer 2 DIDComm Agent Model: DIDComm Messaging Replaces REST/HTTP Messaging

Layer 2 DIDComm Agent Business-to-Business Example (DIDComm Messaging)

The Level 2 DIDComm Agent Model depicted below replaces the REST/HTTP messaging protocol used in the Level 0 REST/HTTP Agent and Level 1 DID Addressable REST/HTTP Agent models with DIDComm Messaging.

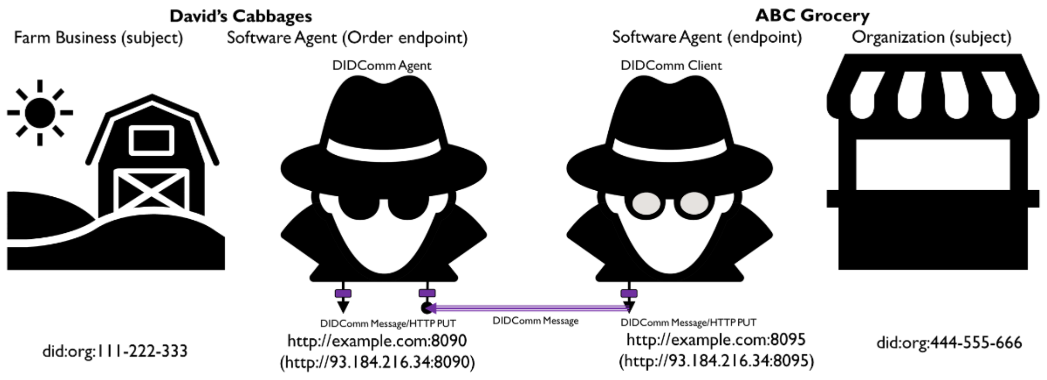


Figure 51. Layer 1 DIDComm Agent Business-to-Business Example

In this scenario, ABC Grocery wants to issue a purchase order for 10 heads of cabbage to David's Cabbages. Each head of cabbage has a value of $1.00. The order is tax-exempt and the shipping method is "pick up". The total value of the purchase order for 10 heads of cabbage is $10.00.

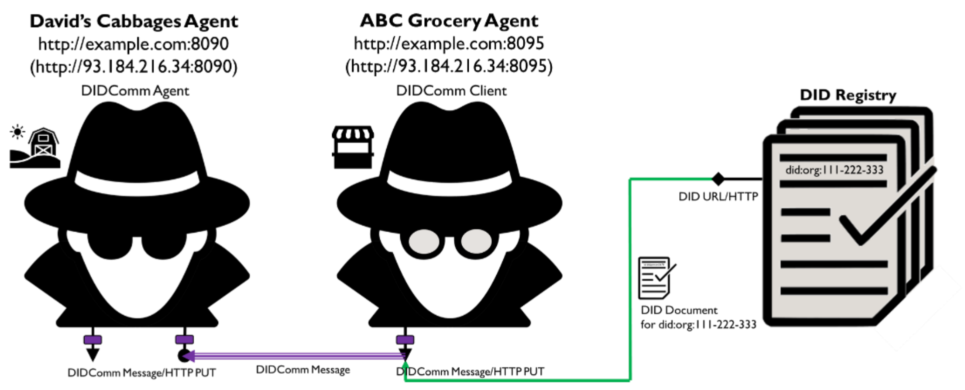


Figure 52. Level 2 DIDComm Agent Model

As a Level 2 DIDComm Agent scenario, the only identifying information that ABC Grocery knows about David's Cabbages is the company's decentralized identifier. However, to transmit the purchase order to David's Cabbages, ABC Grocery needs to determine the Internet address (and port number) for David's Cabbages' DIDComm-enabled software agent. ABC Grocery can accomplish this by resolving David's Cabbages' decentralized identifier against a DID Registry that David's Cabbages is registered with.

Layer 2 DIDComm Agent Model Workflow

The following is an outline of the workflow depicted in the above model. The steps that are new (or changed) in this workflow are highlighted in bold.

1. Verify that Bob's decentralized identifier did:person:5678 is registered on a particular DID Registry.
2. Resolve Bob's decentralized identifier did:person:5678 from the DID Registry to receive a copy of the DID Document associated with this DID.
3. Dereference the "serviceEndpoint" subelement of the appropriate "service" element in the received DID Document
4. Use the value of the "serviceEndpoint" subelement as the Agent's service endpoint Internet address (and port number if specified). If no port number is specified, assume it has a value of 80.
5. Resolve the Agent's service endpoint Internet address and port number to an IP address and port number.
6. Prepare the content of the message to be sent (e.g. plain old character string, JSON serialization of a plain old class object (POCO), etc.).
7. **Outbound Processing: Create a CoreMessage initializing it with:**
    a. **the From DID (Alice's DID),**
    b. **an array of To DIDs (Bob's DID),**
    c. **the Expires attribute, and**
    d. **the message content.**
8. **Instantiate an instance of an HTTPClient**
9. **For every DID in the array of To DIDs, perform Outbound Processing on the CoreMessage and post it to the Receiver Agent according to the following steps:**
    a. **Outbound Processing: Pack a serialized version of the CoreMessage into an EncryptedPackage using the Sender Agent's private key, and the Receiver Agent's public key. The Sender Agent's private key is used to sign the package so that the Receiver**

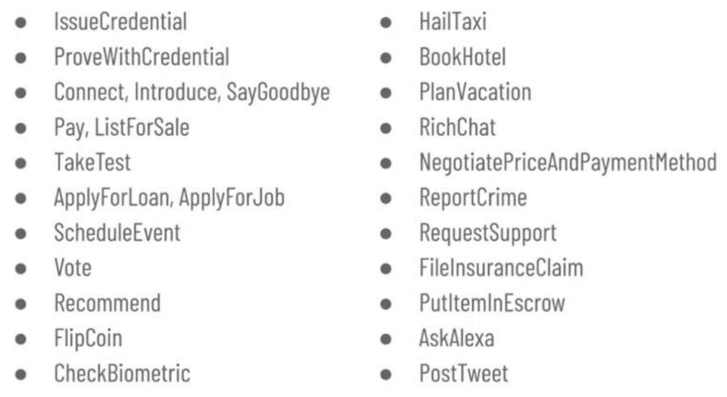
Agent can use the Sender Agent's public key to authenticate the Sender Agent as the expected Sender Agent. The Receiver Agent's public key is used to encrypt the package so that the Receiver Agent can ensure that the package can only be decrypted and read by the intended Receiver Agent (using their private key).

b. **Outbound Processing: Embed the EncryptedPackage in an EncryptedMessage.**
c. **Call the Post method on the HTTPClient instance to send the EncryptedMessage to the resolved IP address and port number of the Receiver Agent as an HTTP request.**
d. **Wait for the (empty) response.**
e. **Check the HTTP error code returned.**

10. **For every Encrypted Message received and accepted by the Receiver Agent, perform Inbound Processing according to the following steps:**
    a. **Inbound Processing: Extract the EncryptedPackage from the EncryptedMessage**
    b. **Inbound Processing: Authenticate the Sender Agent**
    c. **Inbound Processing: Unpack the EncryptedPackage and extract the message content**
    d. **Process the extracted message content**

DIDComm Protocols

Now that steps in the Layer 2 DIDComm Agent Model Workflow have been described for preparing, sending, and receiving a DIDComm Message by a DIDComm Agent, it's appropriate to note that this basic pattern for authenticated, encrypted message passing can be used as the foundation for building virtually any application or system specific protocols. Here's a short list of potential DIDComm Protocols.



- IssueCredential
- ProveWithCredential
- Connect, Introduce, SayGoodbye
- Pay, ListForSale
- TakeTest
- ApplyForLoan, ApplyForJob
- ScheduleEvent
- Vote
- Recommend
- FlipCoin
- CheckBiometric
- HailTaxi
- BookHotel
- PlanVacation
- RichChat
- NegotiatePriceAndPaymentMethod
- ReportCrime
- RequestSupport
- FileInsuranceClaim
- PutItemInEscrow
- AskAlexa
- PostTweet

Figure 53. Potential Application-Level Protocols in DIDComm [Daniel Hardman]

In the DIDComm-ARM, this type of secure, private, and authenticated message passing is referred to as Authenticated Message Exchange (AMX).

LAYER 3 DIDCOMM AGENT WITH VERIFIABLE CREDENTIAL ATTACHMENTS MODEL

The Layer 3 DIDComm Agent with Verifiable Credential Attachments Model builds on the Layer 2 DIDComm Agent Model by enabling DIDComm Message Attachments to be attached to a DIDComm Message. One of the most common types of DIDComm Message Attachments is a Verified Credential (described earlier in this whitepaper).

This is a relatively simple but important extension to the Layer 2 DIDComm Agent model: instead of passing ABC Grocery's purchase order to David's Cabbages as an in-band serialization as a regular DIDComm Message, Layer 3 makes use of a formal DIDComm v2 capability to include attachments as part of a DIDComm Message. In the scenario below, DIDComm v2 Message Attachments are used to attach a Verifiable Credential that wraps a Purchase Order credential.

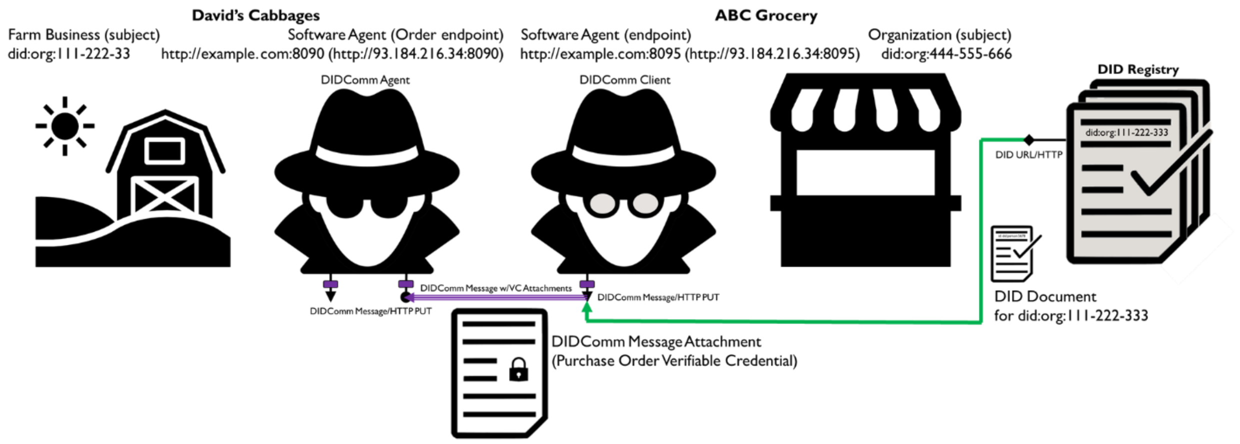


Figure 54. Layer 3 DIDComm Messaging with Verifiable Credential Attachments Model

A more technical visualization of this scenario appears below. In this figure, the role of the DID Registry is highlighted and placed into context with the Verifiable Credential Message Attachment. The point to highlight is that these 2 elements, the DID Registry (and DID Documents) and the use of Verifiable Credentials, are separate and distinct concepts.

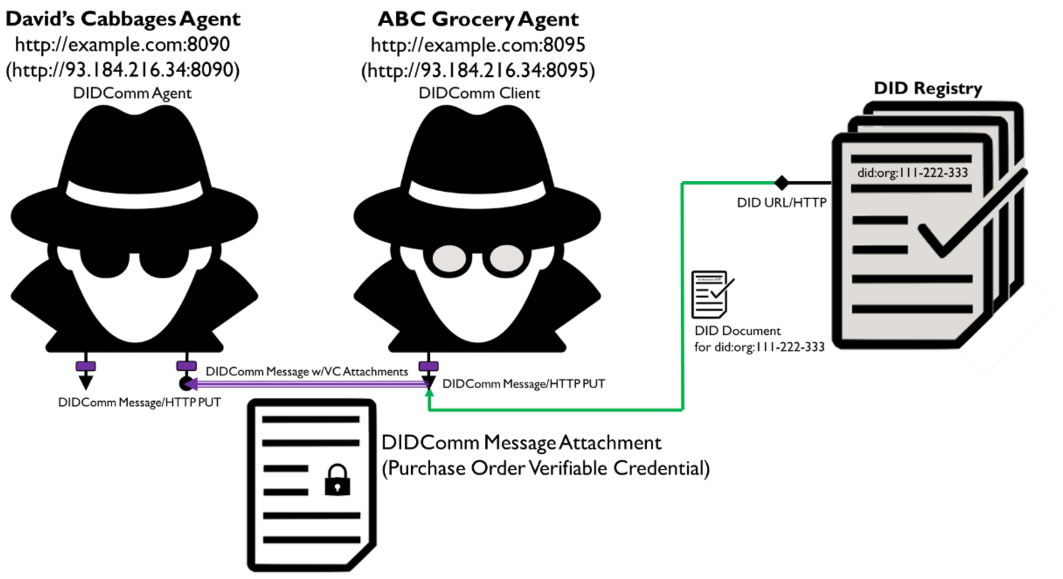


Figure 55. Layer 3 DIDComm Agent with Verifiable Credential Attachments Model

DIDComm Messages vs Verifiable Credentials

When should a solution use secure, private, and authenticated DIDComm Messages (Authenticated Message Exchange) to transport data from one Agent to another Agent vs. using Verifiable Credentials?

When should Verifiable Credentials be used? Verifiable Credentials are ideal for serializing factual or truthful data about a concrete entity; for example, concrete identity information about a person or a company, concrete attributes about a physical object (a house, a boat, a car), a completed business document (e.g. purchase order, invoice, waybill, delivery confirmation, payment receipt, etc.), etc.

Verifiable Credentials should not be used for data or information you wouldn't want to trust 100% as true or factual. Although highly nuanced, this is a principle that underlies virtually every aspect of the efforts to create a decentralized identifier and Verifiable Credential-based ecosystem: truth, privacy, and confidentiality. A preliminary taxonomy of terms that characterizes these concepts can be found in the blog post Facts, Opinions, and Folklore: A Preliminary Taxonomy (https://hyperonomy.com/2022/11/19/facts-opinions-and-folklore-a-preliminary-taxonomy/).

In addition, Verifiable Credentials shouldn't be used when you need to protect the privacy and confidentiality of the claims stored in the credential. The current Verified Credential specification, for example, doesn't contain any provisions for encrypting the credentials (claims) or the entire verified credential itself. Related, there's no enforceable controls over who, what, or where a Verified Credential can be viewed, read, or otherwise inspected. Both of these responsibilities (privacy-and-confidentiality and control over distribution) are unspecified in the current specification and hence, by default, their responsibilities are relegated to other components of the DIDComm-ARM. Options include:

- Using secure data storage (secure containers) for storing Verifiable Credentials (and the data they encapsulate), and
- Attaching Verifiable Credentials to other structures such as DIDComm Messages which are signed and encrypted by default.

If I can't or shouldn't use Verifiable Credentials to represent every kind of entity on the planet, what should I do? When you want secure, trustful, verifiable, authenticated means for transporting data from one agent to another, this is the exact role and purpose of DIDComm Messages[8]. One example is feedback requested and received during a job candidate reference check with each of the candidate's past co-workers. This type of information is: 1) usually obtained as a step in an overall business process (e.g. recruiting) and 2) almost always consists of opinions that difficult to verify …difficult to verify as being 100% true. This is an ideal application for DIDComm Messaging (and DIDComm Agents). The recruiter engages the candidate's reference in a conversation mediated by a pair of DIDComm Agents – perhaps a messaging or chat type of agent. The feedback can be requested and received in real time and/or asynchronously. The interview can be cast as a back-and-forth interview step within the overall recruiting process. The sequence of messages that comprise or make up the dialog of the conversation is naturally going to be authenticated (signed) and kept confidential (encrypted). These messages cannot be shared with anyone else without losing the authentication verifiability. Once the text of the dialog is extracted from a DIDComm Message, it can no longer be reliably or verifiably attributed to the candidate's reference (or to the recruiter).

Authentic Conversations Spectrum

The above discussion gives rise to the concept of the Authentic Conversation Spectrum (illustrated below) based, in part, on the Fact, Opinion, & Folklore Taxonomy[9].

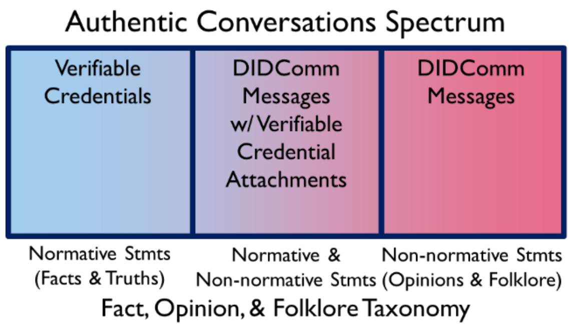

Authentic Conversations Spectrum
Verifiable Credentials
DIDComm Messages w/ Verifiable Credential Attachments
DIDComm Messages
Normative Stmts (Facts & Truths)
Normative & Non-normative Stmts
Non-normative Stmts (Opinions & Folklore)
Fact, Opinion, & Folklore Taxonomy

Figure 56. Authentic Conversations Spectrum

Layer 3 DIDComm Agent with Verifiable Credential Attachments Model Workflow

The following is an outline of the workflow depicted in the above model. The steps that are new (or changed) in this workflow are highlighted in bold.

1. Verify that Bob's decentralized identifier did:person:5678 is registered on a particular DID Registry.
2. Resolve Bob's decentralized identifier did:person:5678 from the DID Registry to receive a copy of the DID Document associated with this DID.

[8] Daniel Hardman, Personal Communication, November 2022.
[9] Facts, Opinions, and Folklore: A Preliminary Taxonomy (https://hyperonomy.com/2022/11/19/facts-opinions-and-folklore-a-preliminary-taxonomy/)

3. Dereference the “serviceEndpoint” subelement of the appropriate “service” element in the received DID Document
4. Use the value of the “serviceEndpoint” subelement as the Agent’s service endpoint Internet address (and port number if specified). If no port number is specified, assume it has a value of 80.
5. Resolve the Agent’s service endpoint Internet address and port number to an IP address and port number.
6. Prepare the content of the message to be sent (e.g. plain old character string, JSON serialization of a plain old class object (POCO), etc.).
7. Outbound Processing: Create a CoreMessage initializing it with:
    a. the From DID (Alice’s DID),
    b. an array of To DIDs (Bob’s DID), and
    c. the Expires attribute plus the message content.
    d. Attach one or more Verifiable Credentials (e.g. ABC Grocery’s purchase order for 10 cabbages) to the Core Message
8. Instantiate an instance of an HTTPClient
9. For every DID in the array of To DIDs, perform Outbound Processing on the CoreMessage and post it to the Receiver Agent
    a. Outbound Processing: Pack a serialized version of the CoreMessage into an EncryptedPackage using the Sender Agent’s private key, and the Receiver Agent’s public key. The Sender Agent’s private key is used to sign the package so that the Receiver Agent can use the Sender Agent’s public key to authenticate that the Sender Agent is the expected Sender Agent. The Receiver Agent’s public key is used to encrypt the package so that the Receiver Agent can ensure that the package can only be decrypted and read by the intended Receiver Agent.
    b. Outbound Processing: Embed the EncryptedPackage in an EncryptedMessage.
    c. Call the Post method on the HTTPClient instance to send the EncryptedMessage to the resolved IP address and port number as an HTTP request.
    d. Wait for the (empty) response.
    e. Check the HTTP error code returned.
10. For every Encrypted Message received and accepted by the Receiver Agent, perform Inbound Processing; then process the extracted message content
    a. Inbound Processing: Extract the EncryptedPackage from the EncryptedMessage
    b. Inbound Processing: Unpack the EncryptedPackage and extract the message content
    c. Process the extracted message content
    d. Detach the Verifiable Credential(s) from the Core Message (e.g. ABC Grocery’s purchase order for 10 cabbages)
    e. Process the detached Verifiable Credential(s) (e.g. submit the Verifiable Credential(s) to a workflow process executing in a workflow engine).

LAYER 4 DIDCOMM AGENT MESH NETWORK MODEL

The Layer 4 DIDComm Agent Mesh Network Model builds on the previous layers of the DIDComm-ARM by replacing the point-to-point DIDComm Message routing between a client and an agent or an agent and a second agent with a network-based message routing solution to support intelligent mesh network routing of DIDComm Messages. This model can be thought of as an extension or elaboration of how Mediators and Relays work with the DIDComm Routing pattern (https://didcomm.org/book/v2/routing).

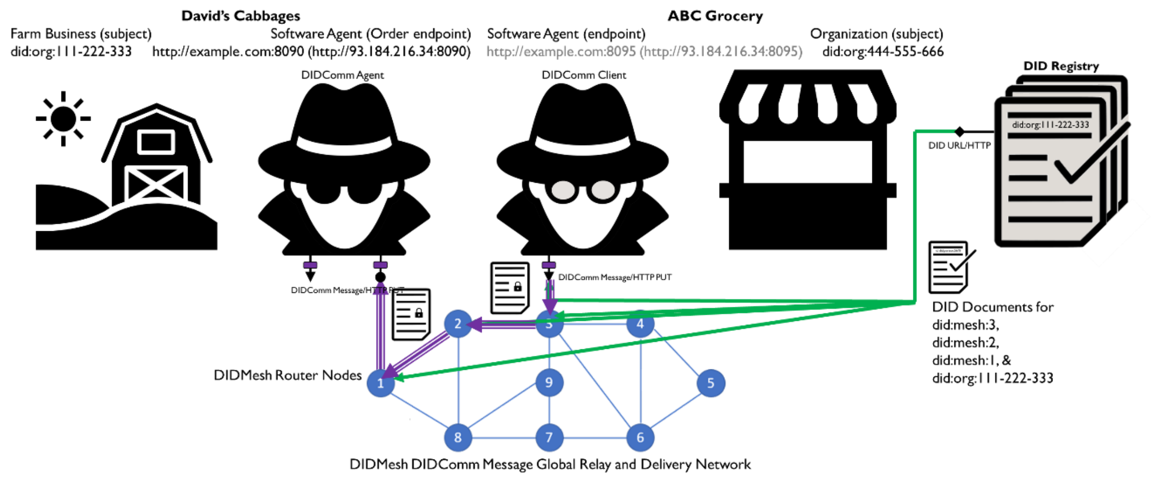


Figure 57. Layer 4 DIDComm Agent Mesh Network Model

In the above figure, the blue dots represent specialized DIDComm Agent Message Router Agents that perform DIDComm Messaging routing operations (similar to the Forward message capability defined in DIDComm Message Routing). Each Router Agent performs various spanning tree calculations to build a local set of message routing (forwarding) tables – analogous to the way Internet network routers maintain local routing tables for routing IP packets from router to router across the Internet.

In the Layer 4 DIDComm Agent Mesh Network model, everything is indexed (identified) using decentralized identifiers: the conventional DIDComm Clients and Agents as well as the Router Agents that make up the DIDComm Agent Mesh Network (aka blue dots). After Router Agent #1 performs a DID-based lookup in its local routing table for the DID of the next Router Agent #2, Router Agent #1 resolves Router #2's DID by querying the DID Registry to retrieve Router Agent #2's DID Document and hence, the Internet address and port number for Router Agent #2's DID Network service endpoint. Router Agent #1 then forwards the in-transit message by sending a request to Router Agent #2's DID Network service endpoint. This process repeats itself until the in-transit DIDComm Message reaches the intended Receiver Agent (e.g. David's Cabbages' DIDComm Agent).

Minimum Viable DIDComm Agent Mesh Networks

The configuration and deployment requirements for a minimum viable DIDComm Agent Mesh depends on the purpose of the network:

- If the network is a “development” network, no DID Network Router Agents might be needed/required.
- If the network is a “test” network, a single DID Network Router Agent is needed to support disconnected agents (offline agents or agents hidden behind a firewall) – if support for disconnected agents is a requirement.
- If the network is a “main” or “production” network, multiple DID Network Router Agents (typically 3 or more) are needed to provide disconnected agent support in the case of a Router Agent failure.

These requirements are over and above the considerations for the ability to support intelligent DID Network Message Routing.

CONCLUSION

If I have seen further, it is by standing on the shoulders of Giants.

[Issac Newton, 1675]

The purpose of this document is to introduce and describe the DIDComm Agent Architecture Reference Model (DIDComm-ARM) and the DIDComm Notation. The latter is a graphical modeling language used to design and model DIDComm Agent-based software systems.

The goals for this document are three-fold:

- Better understanding of the active components of DIDComm Agent-based software systems and how these components rely on and interact with each other,
- Introduce a new graphical modeling language, DIDComm Notation, to aid architects and developers in visualizing new architectures and designs for DIDComm Agent-based software systems, and
- Describe a layered architecture reference model, DIDComm-ARM, to help guide the design and creation of the broadest range of DIDComm Agent-based software systems possible.

The above descriptions of both of DIDComm Notation and the DIDComm Agent Architecture Reference Model (DIDComm-ARM) have delivered the above goals. As a stretch goal, this whitepaper also described an unexpected 5th model: the Layer 4 DIDComm Agent Mesh Network Model[10].

Future Work

Future work on the DIDComm Agent Architecture Reference Model (DIDComm-ARM) can proceed in multiple directions (DIDComm-ARM feature matrix dimensions):

1. Create better sample DIDComm Clients and Agents to provide code examples of all of the capabilities described above. This work is substantially complete and was used to validate the concepts represented here. The code requires a lot of cleanup. The platform and toolset used to create these agents is the DIDComm Superstack (https://www.youtube.com/watch?v=DqTMSRoSFpA&list=PLU-rWqHm5p444nGB7nSvvpRencijFOOOd&index=1). The technology stack used for these sample applications includes: C#/.NET/Visual Studio, Microsoft Graph Engine code generator, and the Trinsic OKAPI libraries.
   The code for the examples for each later of the DIDComm-ARM will be published here: https://github.com/mwherman2000/web7-didcomm-arm-samples.
2. Expand the DIDComm-ARM feature matrix to support additional DIDComm Message transports:
   a. gRPC
   b. libp2p
   c. Tor

[10] Thank you to the outreach of the folks at Veramo Labs (https://veramo.io/) for inspiring the addition of the Layer 5 DIDComm Agent Mesh Network Model. At the time of writing, it's unknown how similar (or dissimilar) the Layer 5 model is with respect to what Veramo is planning. Currently, these are two completely independent efforts.

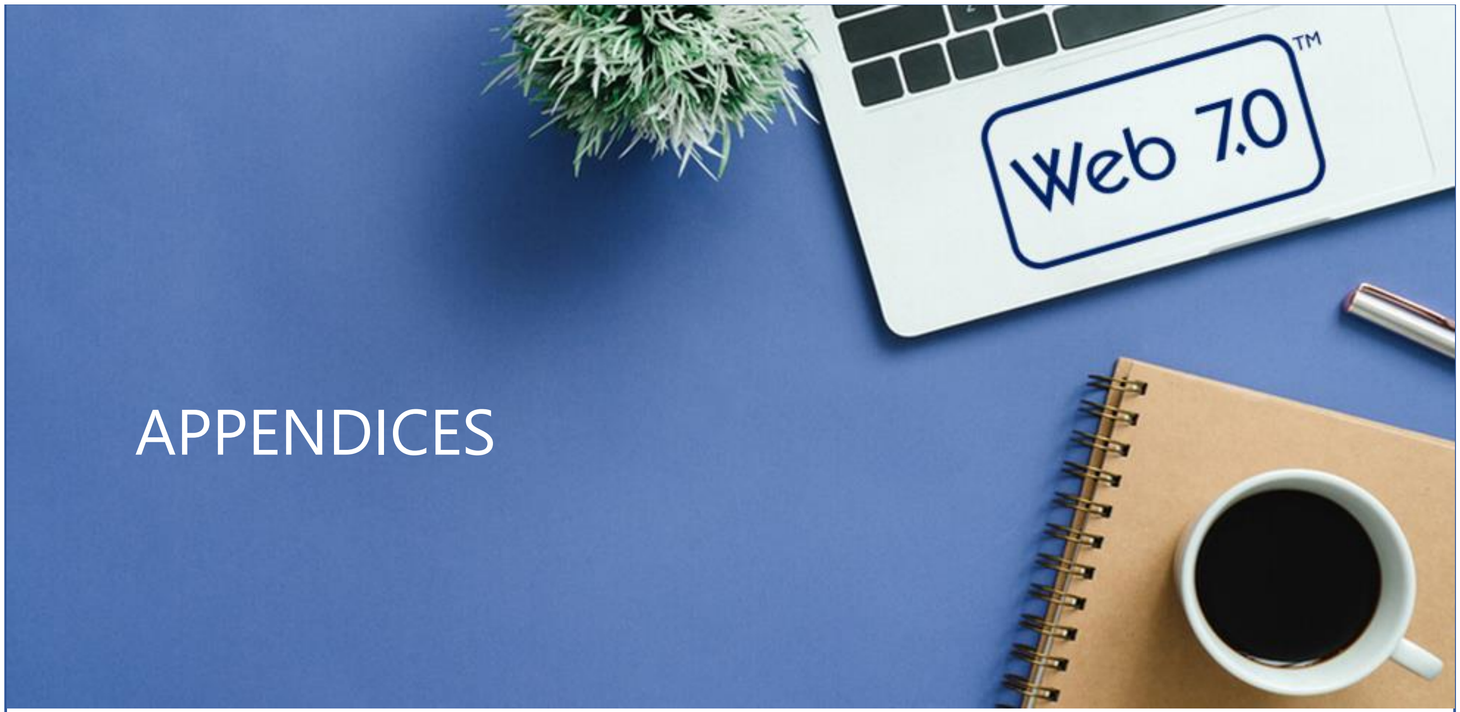
3. Expand the feature matrix to support new as well as existing DIDComm Protocols:
    a. Purchasing/Supply Chain
4. Expand the feature matrix to support disconnected DIDComm Agents
    a. DIDComm Relay Server

Acknowledgments

The responsibility for any errors or omissions lies with the author(s) alone.

The author would like to thank the following for their inspiration:

- Daniel Hardman, CTO, Provenant
- Dr. Kellogg S. Booth, UBC
- Members of the DIDComm Working Group of the Decentralized Identity Foundation (DIF)
- Members of the Veramo community on Discord and LinkedIn
    - Nick Reynolds
    - Alen Horvat



APPENDIX A – LAYER 5 DIDCOMM USER AGENT MODEL

A late addition to the current version of the DIDComm-ARM is the Layer 5 DIDComm User Agent Model. The description of Layer 5 is not included in the main body of this whitepaper. A description has been included in this appendix.

The purpose of Layer 5 is to introduce the concept of a DIDComm User Agent to the DIDComm Notation and the DIDComm-ARM. The Layer 5 DIDComm User Agent Model adds the ability to incorporate specialized DIDComm Agents designed to interact with human actors.

Below is the revised DIDComm-ARM Feature Matrix highlighting the addition of the Layer 5 DIDComm User Agent Model.

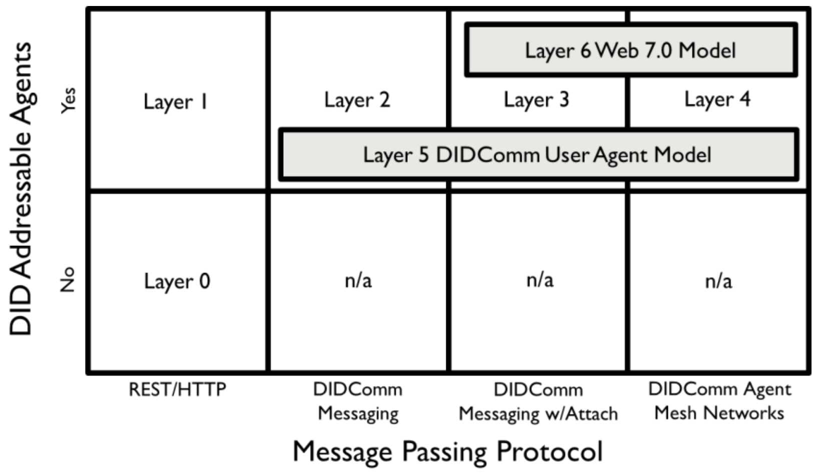


Figure 58. DIDComm-ARM Feature Matrix

Even though the Layer 3 DIDComm Agent with Verifiable Credential Attachments Model and the Layer 4 DIDComm Agent Mesh Network Model are separate and distinct from one another, they can be composed into a single model to suit the needs of the decentralized identifier-based software system you want to build.

The former is characterized by direct point-to-point (or agent-to-agent) message passing while the latter incorporates intelligent routing of DIDComm Messages between agents using a mesh network topology. For symmetry and simplicity, the nodes in the Layer 4 DIDComm Mesh Network Model are implemented as DIDComm Agents (DIDComm Agent Routers, analogous to routers in an IP-based communications network).

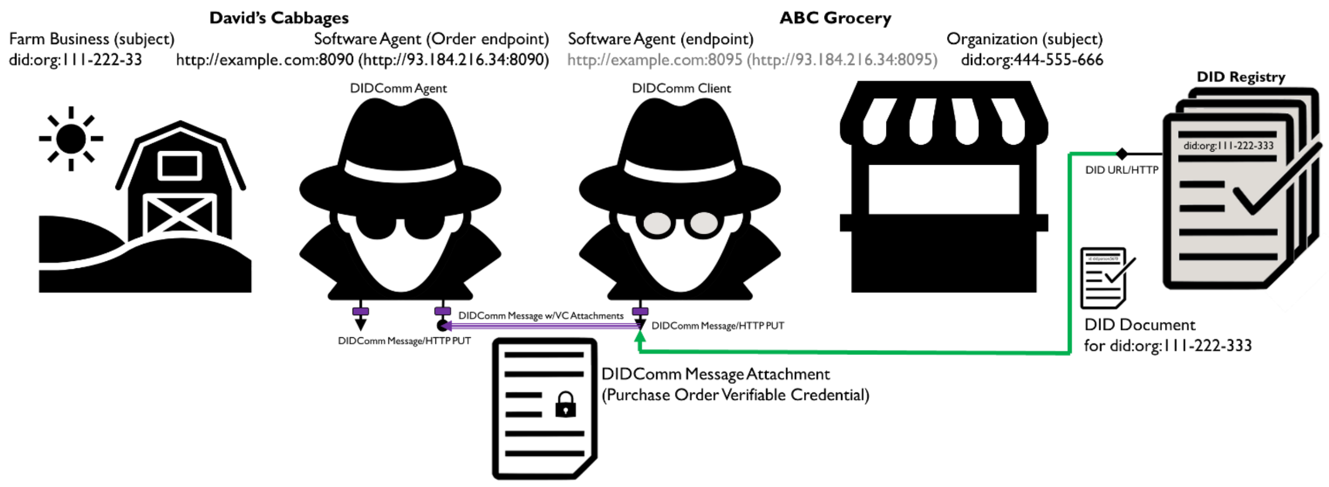


Figure 59. Layer 3 DIDComm Agent w/Verifiable Credential Attachments Model

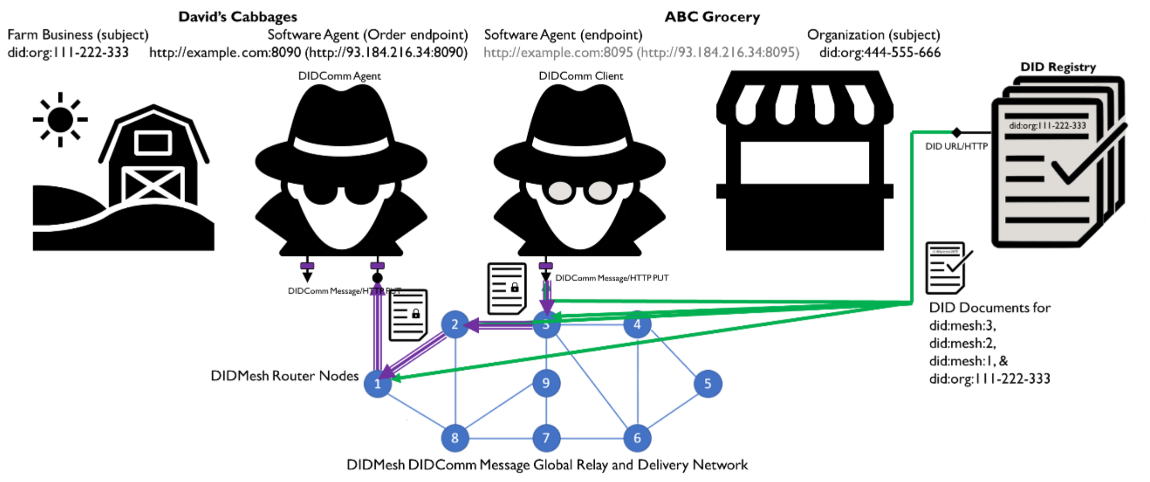
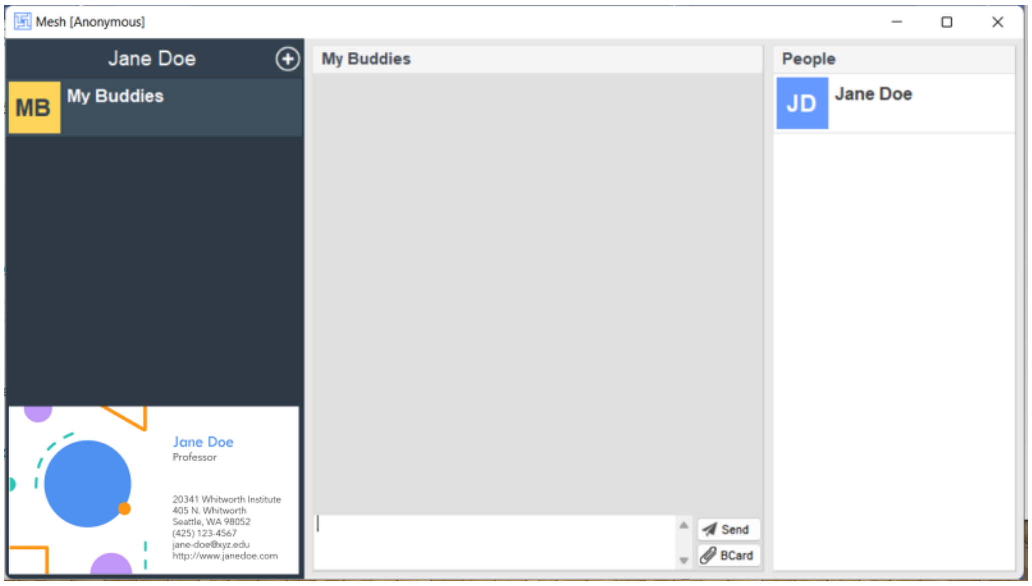


Figure 60. Layer 4 DIDComm Agent Mesh Network Model

The Layer 5 DIDComm User Agent Model starts with either of the above 2 models and adds the ability to incorporate specialized DIDComm Agents designed to interact with human actors; for example, the following chat application that supports group collaboration using both text messaging as well as file transfers. The latter includes the transport of Verifiable Credentials such as Business Cards based on the Person schema example from Schema.org. A sample Business Card is illustrated in the following screenshot.



Jane Doe
Professor
20341 Whitworth Institute
405 N. Whitworth
Seattle, WA 98052
(425) 123-4567
jane-doe@xyz.edu
http://www.janedoe.com

Figure 61. Layer 3 DIDComm Agent Chat Sample using Schema.org Person Verifiable Credential Attachments

The following figure shows where a DIDComm User Agent fits into the overall DIDComm Network model.

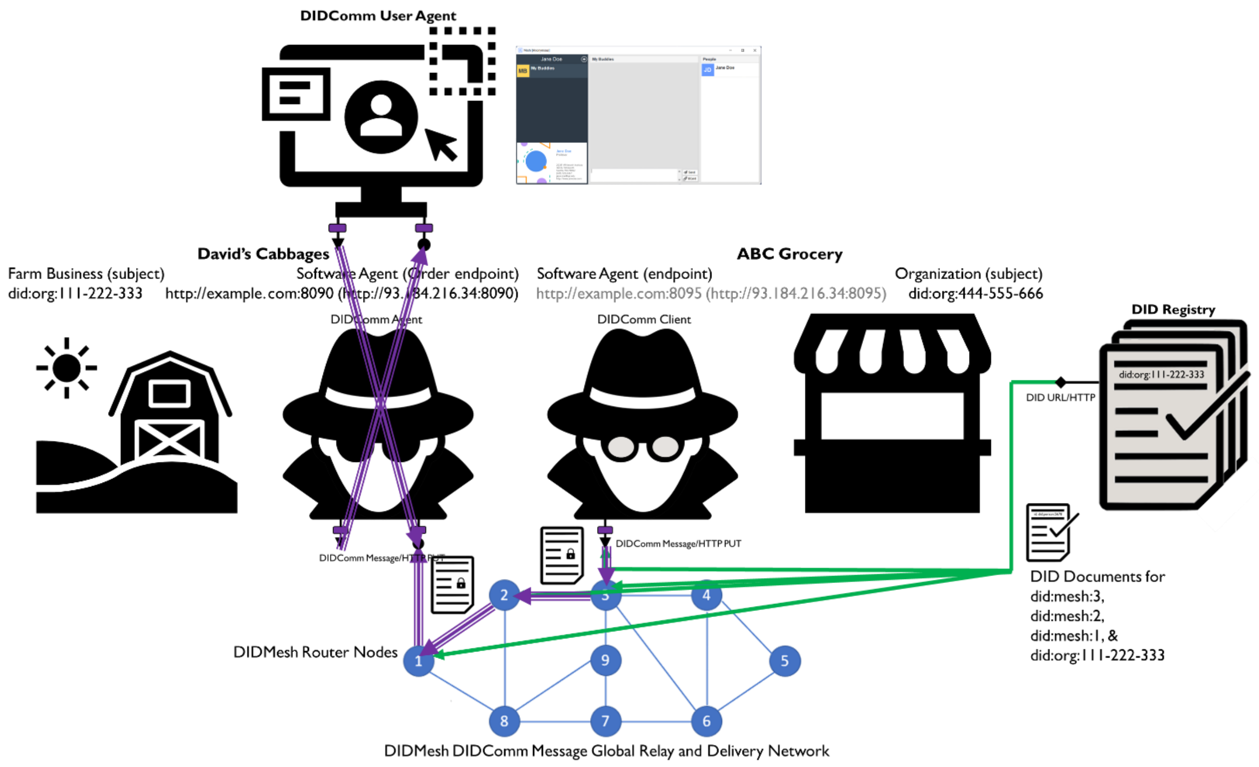


Figure 62. Layer 5 DIDComm User Agent Model

The complete model for the Layer 5 DIDComm User Agent Model is becoming quite elaborate:

- The DIDComm User Agent only exchanges messages with the DIDComm Agent it is paired with. Outbound messages are sent from the User Agent's Origin interface to the Endpoint interface of the DIDComm Agent it is paired with. Outbound messages from the DIDComm Agent it is paired with are received by the User Agent's Endpoint interface.
- All communication with other (non-user agent) DIDComm Agents is performed using direct, point-to-point message passing (the Layer 3 model) or through message routing in the DIDComm Mesh Network model (the Layer 4 model).

As an example, let's apply this model to the Chat DIDComm User Agent application:

- When a user enters some text and clicks Send, the intended Receiver Agents of the text message are all of the participants in the collaboration group. More technically, the text message is packaged up (created, signed, and encrypted) and sent to the DIDComm Agent the Chat application is paired with.
- The paired DIDComm Agent then routes a copy of the message from the User Agent to: a) itself and b) each of the DIDComm Agents that are paired with the collaborators who are in the group conversation.
- One role that the paired DIDComm Agent takes on is being the manager for a secure data store (SDS) where the text messages (DIDComm Messages) are tracked and stored.

- When the paired DIDComm Agent receives DIDComm Messages from other paired DIDComm Agents (containing text messages from other users), the paired DIDComm Agent also stored these in its SDS storage.
- Whenever there is a change in the list of messages in the paired DIDComm Agent's SDS storage, the paired agent sends a notification DIDComm Message to the Chat DIDComm User Agent it is paired with. The Chat DIDComm User Agent then updates the scrolling list of chat messages in its user interface with the new changes. These changes include any text messages the user just sent by clicking the Send button as well as any text messages received from other DIDComm Agents.

The Chat DIDComm User Agent also supports file transfers between the Chat DIDComm User Agent and its paired DIDComm Agent using DIDComm Messages and DIDComm Message Attachments. Similarly, because these are simply conformant DIDComm Messages and DIDComm Message Attachments, these messages can also be automatically exchanged with other DIDComm Agents. Any type of file can be transferred using this capability – any type of file including Verifiable Credentials such as a Business Card that conforms with the Person schema example from Schema.org.

On receipt of an inbound file (as a DIDComm Message Attachment to a DIDComm Message), a DIDComm User Agent can process these files any way that it chooses to. If, after inspecting a DIDComm Message Attachment, it determines that it is a Verifiable Credential that is compatible with the Schema.org Person schema, the DIDComm User Agent can render the Person Verifiable Credential as an actual business card (as shown above) or submit it to a workflow engine component with the DIDComm Agent for subsequent processing.

Lastly, it is not the role of the DIDComm User Agent to store and curate any DIDComm Messages, DIDComm Message Attachments, or Verifiable Credentials beyond the length of the user session. It is the responsibility of the DIDComm Agent the DIDComm User Agent is paired with to manage the long-term storage and curation of these assets.

APPENDIX B – LAYER 6 WEB 7.0 DIDCOMM AGENT ARCHITECTURE MODEL

A late addition to the current version of the DIDComm-ARM is the Layer 6 Web 7.0 DIDComm Agent Architecture Model (found here in Appendix B).

Below is the revised DIDComm-ARM Feature Matrix highlighting the addition of the Layer 6 Web 7.0 DIDComm Agent Architecture Model.

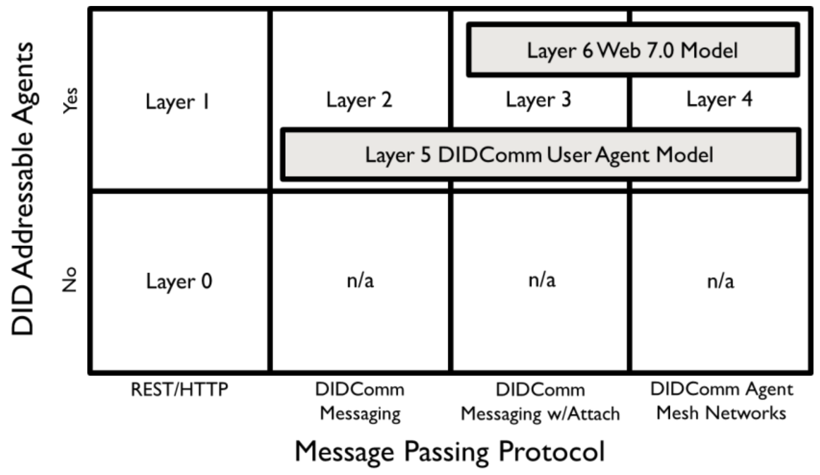


Figure 63. DIDComm-ARM Feature Matrix

The overarching objective of Web 7.0 is to connect individuals (people and organizations) to each other (via their trusted person agents).

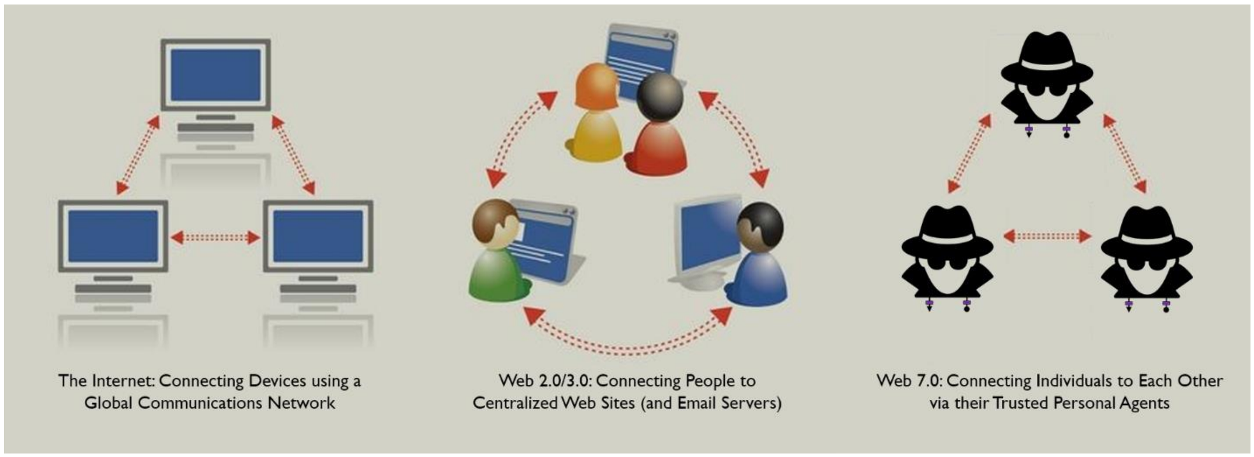

The Internet: Connecting Devices using a Global Communications Network
Web 2.0/3.0: Connecting People to Centralized Web Sites (and Email Servers)
Web 7.0: Connecting Individuals to Each Other via their Trusted Personal Agents

Figure 64. The Internet vs. Web 2.0/3.0 vs. Web 7.0[11]

[11] This diagram is an extension of a diagram that contrasted the Internet and WWW produced by the Computer History Museum.

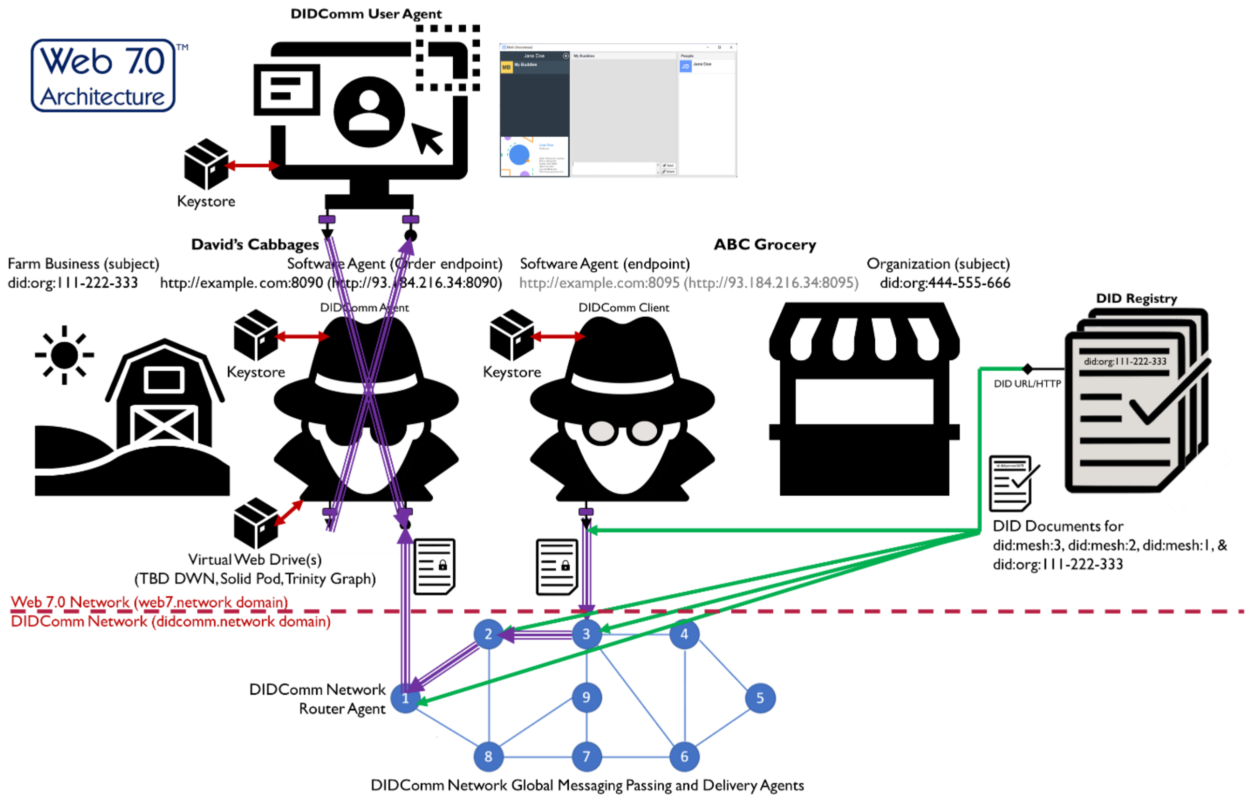
The Web 7.0 connections (and the information that flows over them) are safe, secure, and private.

The defining features of the Layer 6 Web 7.0 DIDComm Agent Architecture Model include:

- The separation between the Web 7.0 Network of agents and the DIDComm Network of router (relay) agents to support reliable message passing for connected and disconnected agents, and
- The ability to attach multiple Virtual Web Drives (VWD) to a single DIDComm Agent.

The most frequent scenario is for VWDs to be attached to DIDComm Agents that are paired with DIDComm User Agents (and not the DIDComm User Agents themselves). VWDs may be attached to other DIDComm Agents that have special roles (e.g. DID Network Router Agents).

The Web 7.0 Network runs on top of the DIDComm Network is manner that is analogous to the way the World Wide Web (WWW) runs today as an application on top of the Internet global communications network.



Web 7.0 Architecture TM
DIDComm User Agent
Keystore
David's Cabbages
Farm Business (subject)
did:org:111-222-333
Software Agent (Order endpoint)
http://example.com:8090 (http://93.184.216.34:8090)
DIDComm Agent
Keystore
Virtual Web Drive(s)
(TBD DWN, Solid Pod, Trinity Graph)
ABC Grocery
Software Agent (endpoint)
http://example.com:8095 (http://93.184.216.34:8095)
DIDComm Client
Keystore
Organization (subject)
did:org:444-555-666
DID Registry
did:org:111-222-333
DID URL/HTTP
DID Documents for
did:mesh:3, did:mesh:2, did:mesh:1, &
did:org:111-222-333
Web 7.0 Network (web7.network domain)
DIDComm Network (didcomm.network domain)
DIDComm Network
Router Agent
1
2
3
4
5
6
7
8
9
DIDComm Network Global Messaging Passing and Delivery Agents

Figure 65. Layer 6 Web 7.0 DIDComm Agent Architecture Model

Web 7.0 and DIDComm Networks

The Web 7.0 Network runs on top of the DIDComm Network is manner that is analogous to the way the World Wide Web (WWW) runs today as an application on top of the Internet global communications network.

The Web 7.0 Network includes all the DIDComm User Agents and their paired DIDComm Agents. The Web 7.0 Network also includes specialized DIDComm Agents such as a DIDRegistry Gateway Agent or similar data and information integration gateway agents. The exceptions include the DIDComm Network Router Agents that are the sole agent roles in the DIDComm Network. DIDComm Network Router Agents are responsible for supporting reliable message passing (DIDComm Message routing and delivery) for connected and disconnected agents in the Web 7.0 Network as well as fellow Router Agents in the DIDComm Network.

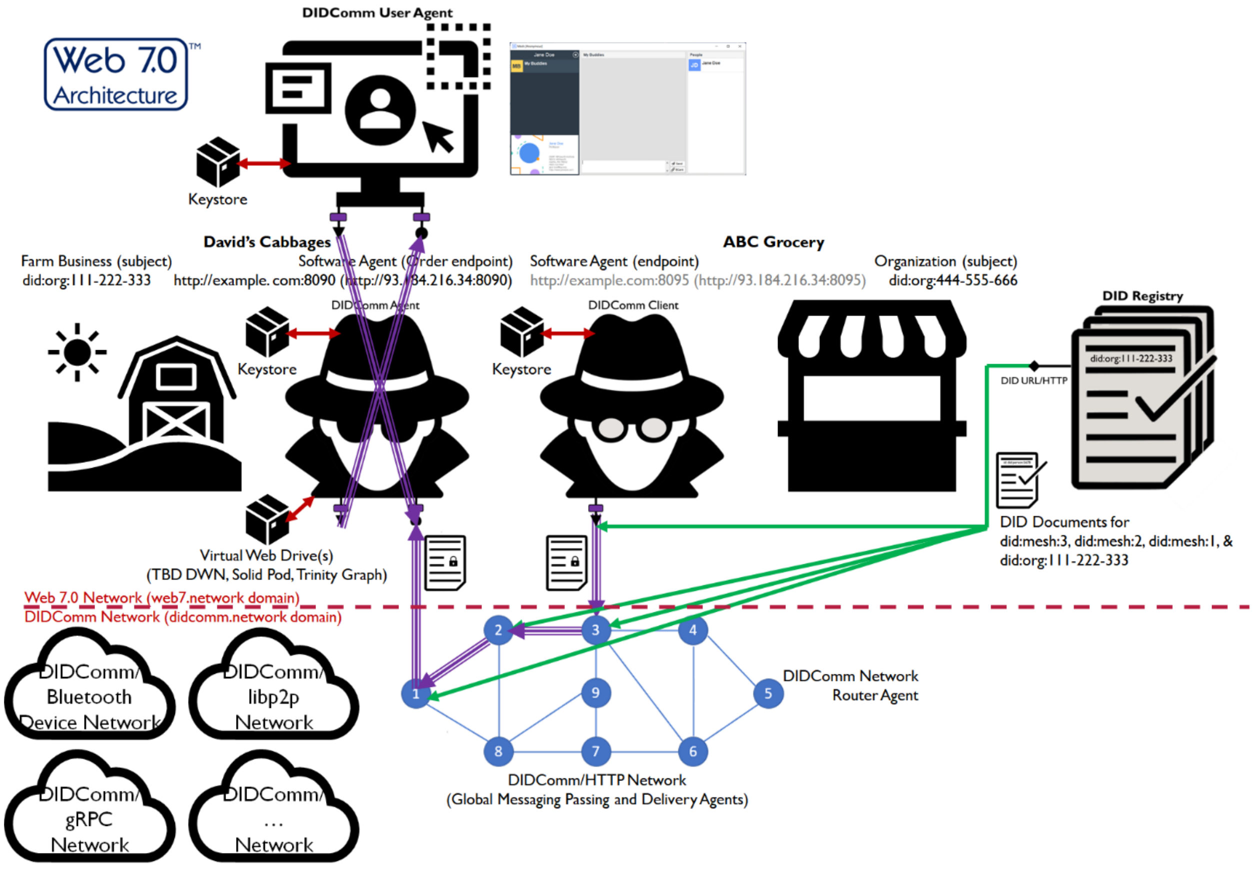

DIDComm User Agent
Web 7.0 Architecture
Keystore
David's Cabbages
ABC Grocery
Farm Business (subject)
did:org:111-222-333
Software Agent (Order endpoint)
http://example.com:8090 (http://93.184.216.34:8090)
Software Agent (endpoint)
http://example.com:8095 (http://93.184.216.34:8095)
Organization (subject)
did:org:444-555-666
DID Registry
DIDComm Agent
DIDComm Client
Keystore
Keystore
DID URL/HTTP
did:org:111-222-333
DID Documents for did:mesh:3, did:mesh:2, did:mesh:1, & did:org:111-222-333
Virtual Web Drive(s)
(TBD DWN, Solid Pod, Trinity Graph)
Web 7.0 Network (web7.network domain)
DIDComm Network (didcomm.network domain)
DIDComm/ Bluetooth Device Network
DIDComm/ libp2p Network
DIDComm/ gRPC Network
DIDComm/ … Network
DIDComm Network Router Agent
DIDComm/HTTP Network
(Global Messaging Passing and Delivery Agents)
1
2
3
4
5
6
7
8
9

Figure 66. Layer 6 Web 7.0 DIDComm Agent Architecture Model: Multiple DIDComm Networks

The above figure illustrates how Web 7.0 embraces multiple DIDComm Networks running on multiple network transport protocols (e.g. DIDComm/Bluetooth, DIDComm/libp2p, DIDComm/gRPC, etc.). DIDComm Agents can send and receive DIDComm Messages of any of these individual networks. It is also possible to design and implement DIDComm Agent-based network gateways for inter-network message routing and delivery.

Virtual Web Drives

The Virtual Web Drive abstraction enables a variety of secure data storage (SDS) technologies to be attached to a DIDComm Agent. Examples of SDS technologies that can be attached to and accessed by a DIDComm Agent include:

- TBD Decentralized Web Nodes (DWNs)
- Solid Pods (Pods)
- Trinity Graph

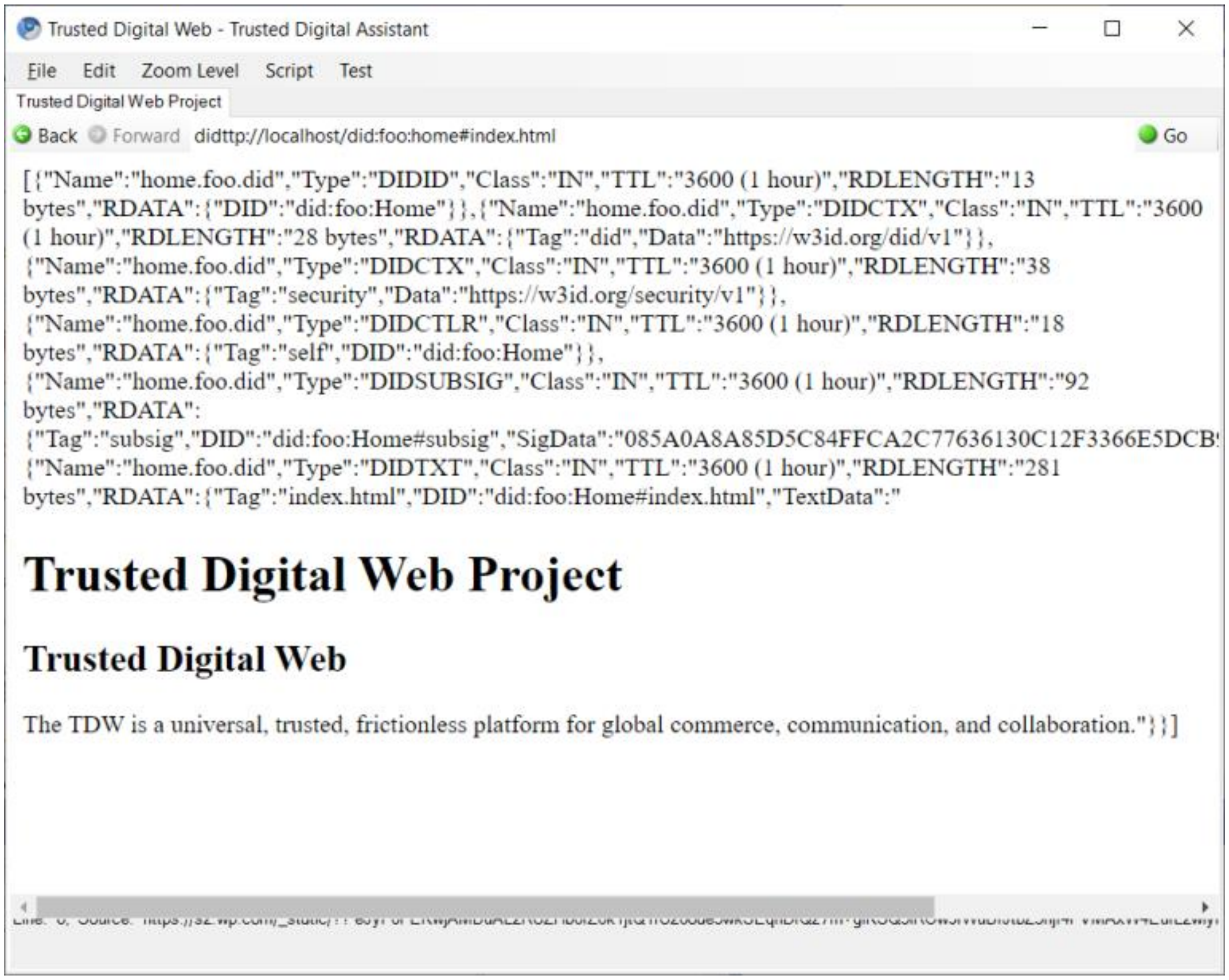
VWDs are used for storing:

- DIDComm Messages
- DIDComm Message Attachments
- Verifiable Credentials received as DIDComm Message Attachments
- Verifiable Credentials issued by the paired DIDComm Agent

Initially, a particular VWD will have an exclusive 1:1 relationship with a paired DIDComm Agent. In the future, it may be possible to have multiple DIDComm Agents access a particular VWD; initially, in read-only mode; then append mode; then update mode.

First Web 7.0 Web Page

Although it wasn't called Web 7.0 on December 3, 2019 (three years ago today) and it didn't use DIDComm as a transport or a virtual web drive, the demonstration on that day of the Trusted Digital Assistant rendering the equivalent of a web page verifiable credential is the first example of a Web 7.0 web page. Here's a screenshot of what, today, would be considered a prototype of a Web 7.0 web browser (https://hyperonomy.com/2019/12/03/trusted-digital-web-first-trusted-web-page-delivered-today-dec-3-2019/). Note use of DIDs as a web page addresses.


Trusted Digital Web - Trusted Digital Assistant
File
Edit
Zoom Level
Script
Test
Trusted Digital Web Project
Back
Forward
Go

didttp://localhost/did:foo:home#index.html

```
[{"Name":"home.foo.did","Type":"DIDID","Class":"IN","TTL":"3600 (1 hour)","RDLENGTH":"13
bytes","RDATA":{"DID":"did:foo:Home"}},{"Name":"home.foo.did","Type":"DIDCTX","Class":"IN","TTL":"3600
(1 hour)","RDLENGTH":"28 bytes","RDATA":{"Tag":"did","Data":"https://w3id.org/did/v1"}},
{"Name":"home.foo.did","Type":"DIDCTX","Class":"IN","TTL":"3600 (1 hour)","RDLENGTH":"38
bytes","RDATA":{"Tag":"security","Data":"https://w3id.org/security/v1"}},
{"Name":"home.foo.did","Type":"DIDCTLR","Class":"IN","TTL":"3600 (1 hour)","RDLENGTH":"18
bytes","RDATA":{"Tag":"self","DID":"did:foo:Home"}},
{"Name":"home.foo.did","Type":"DIDSUBSIG","Class":"IN","TTL":"3600 (1 hour)","RDLENGTH":"92
bytes","RDATA":
{"Tag":"subsig","DID":"did:foo:Home#subsig","SigData":"085A0A8A85D5C84FFCA2C77636130C12F3366E5DCB
{"Name":"home.foo.did","Type":"DIDTXT","Class":"IN","TTL":"3600 (1 hour)","RDLENGTH":"281
bytes","RDATA":{"Tag":"index.html","DID":"did:foo:Home#index.html","TextData":"
```

Trusted Digital Web Project

Trusted Digital Web

The TDW is a universal, trusted, frictionless platform for global commerce, communication, and collaboration."}}]

Figure 67. First Web 7.0 Web Page

APPENDIX C – ABOUT THE AUTHOR

Michael Herman

Self-Sovereign Blockchain Architect, Developer and Futurist
Trusted Digital Web
Hyperonomy Digital Identity Lab
Parallelspace Corporation

The author is a seasoned software professional with more than 45 years of experience working for a range of software companies large and small; including Microsoft and IBM. His experience includes several years working as an enterprise architect certified in the use of the ArchiMate enterprise architecture modeling language and the TOGAF enterprise architecture framework. As a professional writer, the author specializes in making new technologies "easier to learn, easier to understand, and easier for you to explain to others".

The author has more than 20 years of experience architecting and implementing decentralized software solutions beginning with the Groove Workspace platform from Groove Networks, Inc. acquired by Microsoft Corporation in 2005 (https://en.wikipedia.org/wiki/Groove_Networks), the `ddtp` protocol handler for Microsoft Exchange Server and Microsoft Outlook, and Syntergy Replicator (https://www.youtube.com/watch?v=o_YjDTVwuQg).

In his work with the decentralized identity community, the author is a named contributor to the Decentralized Identifiers (DIDs) v1.0 W3C Recommendation (https://www.w3.org/TR/did-core/#acknowledgements).

The author has produced the following detailed Architecture Reference Models (ARMS) for many different technology platforms.

1. Trusted Digital Web: 8-Layer Architecture Reference Model (TDW-ARM) (https://hyperonomy.com/2021/06/28/trusted-digital-web-8-layer-architecture-reference-model-tdw-arm/)
2. Aries RFC 0004: Agents: Agent Categorization (https://github.com/hyperledger/aries-rfcs/blob/main/concepts/0004-agents/README.md#categorizing-agents)
3. Secure Data Storage Working Group (sds-wg) Confidential Storage (CS): Functional Architecture Reference Models (CS-FARMs) 0.36 (whitepaper placeholder): Sample Diagrams (https://hyperonomy.com/2021/02/10/secure-data-storage-working-group-sds-wg-confidential-storage-cs-functional-architecture-reference-models-cs-farms/)
4. Hyperledger Indy/Sovrin/DID Comprehensive Architecture Reference Model (INDY ARM) (https://github.com/mwherman2000/indy-arm/blob/master/README.md)
5. INDY ARM Interactive Explorer (https://mwherman2000.github.io/indy-arm/)
6. HyperLedger Indy Getting Started Guide for Enterprise Architects and Developers (INDY GSG-EA) (https://github.com/mwherman2000/indy-gsg-ea/blob/master/python/doc/getting_started-enterpise.md)
7. Sovrin Governance Framework Comprehensive Architecture Reference Model (SOVRIN ARM) (https://github.com/mwherman2000/sovrin-arm/blob/master/README.md)
8. Sovrin ARM Interactive Explorer (https://mwherman2000.github.io/sovrin-arm/)

9. The Structured Credential Model (https://www.youtube.com/watch?v=9RLYS7Xvabc&list=PLU-rWqHm5p445PbGKoc9dnlsYcWZ8X9VX&index=2)
10. The Verifiable Economy Architecture Reference Model (VE-ARM): Fully Decentralized Object (FDO) Model (https://hyperonomy.com/2021/04/26/the-verifiable-economy-architecture-reference-model-ve-arm-fdo/)
11. Microsoft Azure Stack POC Architecture Reference Model (ARM) (https://hyperonomy.com/2016/04/29/microsoft-azure-stack-poc-architecture-archimate-model-version-1-0-1-april-29-2016/)
12. ARM Models for Model-Driven LOB apps: SharePoint 2013/SharePoint 2016 (https://hyperonomy.com/2016/06/15/arms-for-metadata-driven-lob-apps-sharepoint-2013sharepoint-2016/)

DIDComm Super Stack

The author is the architect and developer responsible for creating the DIDComm Super Stack development platform for .NET developers (https://www.youtube.com/watch?v=DqTMSRoSFpA&list=PLU-rWqHm5p444nGB7nSvvpRencijFOOOd&index=1).

DID/DNS

DID/DNS was implemented by the author by adding a range of new DNS Resource Record types to an industry-standard, specifications-compliant Domain Name Service (DNS) Server. One implementation is described here: https://hyperonomy.com/2019/12/03/trusted-digital-web-first-trusted-web-page-delivered-today-dec-3-2019/. The Trusted Digital Web Data Registry can be transparently deployed as a forwarding DNS server (logically in front of any existing DNS service). The open-source repository for this implementation can be found on GitHub: https://github.com/mwherman2000/DnsServer. The DNS (Domain Name Service) specification can be found here: https://www.ietf.org/rfc/rfc1035.txt. A primer on the DNS protocol can be found here: https://hyperonomy.com/2019/01/02/dns-domain-name-service-a-detailed-high-level-overview/.

DIDLANG/HTTP

DIDLANG/HTTP is an experimental domain-specific language (DSL) created and implemented by author to support full CRUD capabilities against DID Documents in DID Registries and DID Objects managed by DID Agents. Based on the concepts of DID Indirection and DID Coercion, DIDLANG is the equivalent of "SQL for DIDs, DID Documents, and DID Objects". One open-source implementation is described here: https://github.com/mwherman2000/BlueToqueTools#didlang-language-command-line-interpreter-for-did-method-namespaces-did-identifiers-did-documents-did-agent-service-endpoints-did-agent-servers-did-agent-clusters-and-did-objects-version-04. A demonstration of the current DIDLANG processor can found here: https://www.youtube.com/playlist?list=PLU-rWqHm5p45RKAAF8bleTuPNI5cBE3gP.

APPENDIX D – COPYRIGHT, TRADMARKS, AND LICENSES

Copyright

This whitepaper is:

Copyright (c) 2022 Michael Herman (Alberta, Canada) - Creative Commons Attribution-ShareAlike 4.0 International Public License

Trademarks

The following graphic logo trademarks (and other similar trademarks) are the property of Michael Herman, Alberta, Canada. The phrase "Web 7.0" is not trademarked. Only the graphic logo trademarks are protected.

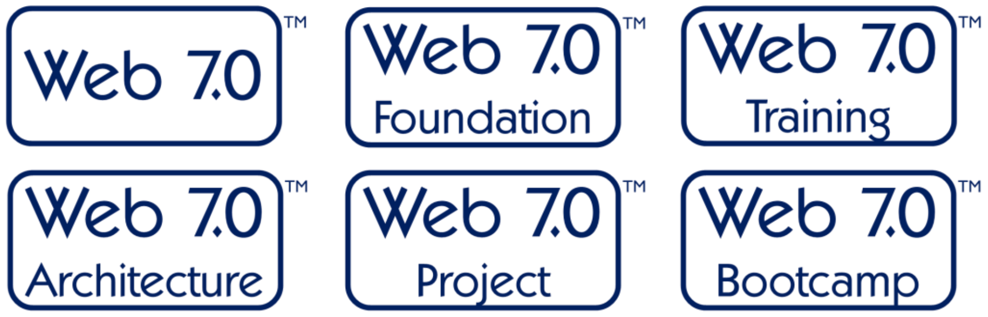


Figure 68. Web 7.0 Graphic Logo Trademarks

Licenses

Creative Commons Attribution-ShareAlike 4.0 International Public License

Creative Commons Attribution-ShareAlike 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-ShareAlike 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

    A. reproduce and Share the Licensed Material, in whole or in part; and

    B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

    A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

    B. Additional offer from the Licensor – Adapted Material. Every recipient of Adapted Material from You automatically receives an offer from the Licensor to exercise the Licensed Rights in the Adapted Material under the conditions of the Adapter's License You apply.

    C. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. **Other rights**.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the