Ecma/TC56/2025/019

2nd Draft Technical Report ECMA TR/XXX

1st Edition / July 2025

Overview of the Natural Language Interaction Protocol (NLIP)

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

is the registered trademark of Ecma International

COPYRIGHT PROTECTED DOCUMENT

Contents

Page

Introduction

The advent of large language models (LLMs) has made an interactive natural language interaction feasible between machines in a manner that did not exist before. An implication is that a natural language interface can replace many mobile applications that are used today.

Just like the advent of the browser in 1990s simplified technology by replacing a plethora of client-side applications with a single standard application, a common natural language interaction protocol can potentially replace the plethora of mobile applications that exists today, providing a universal application layer protocol. Convergence to a universal application layer protocol would bring significant benefits to all segments of society – consumers can use a single application for various interactions, businesses will have a simpler maintenance burden for their IT infrastructure, and integration among different businesses can be streamlined.

Ecma Technical Committee 56 is defining this Standard. Two Standards define the standard protocol called Natural Language Interaction Protocol (NLIP) and its binding over HTTPS/REST.

In this document, the motivation, requirements and design philosophy behind the NLIP Standards are provided. Some use-cases and sample interactions among different parties for various common scenarios are also documented.

This Ecma Technical Report was developed by Technical Committee 56 and was adopted by the General Assembly of <month> <year>.

Overview of the Natural Language Interactive Protocol (NLIP)

1 Scope

This Technical Report provides a supporting documentation for the NLIP specifications and binding document. It includes the motivation, the requirements of the protocol, and examples of sample exchanges. This is a normative document and does not define a standard. In the case of a conflict between this document and the NLIP standards, the standards document take precedence.

2 References

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

3 Abbreviations

ALG Application Layer Gateway

API Application Protocol Interface

4 Ecma TC56's vision

This clause defines Ecma TC56's vision for the future that a protocol like NLIP can enable.

TC56 envisions a future in which there is a single application on the phone of consumers, which can interact with business services provided by various businesses or organizations. We provide some instances where the current environment causes hardships for users, and where we envision a single application in operation to provide significant benefits to users and service providers.

At the present time, almost every major city has a public transportation system that publishes its own mobile application for the benefits of its riders. The application provides many capabilities including an easy view into train timetables and status, an ability to view and buy ride tickets/passes, and an ability to find how to go from point A to point B. At the same time, every city provides its own application, which requires installing an application for every city. This imposes a significant hassle for any individual who travels to multiple cities.

Consider the case of a traveling trade-person who is on a tour to attend conferences, trade shows, and to meet with clients. The person needs to install the train/public transit/taxi application for each of the cities in order to determine how to move around conveniently. If the traveller is not a frequent visitor to the cities, the need to install the plethora of apps is a significant burden.

The same traveller is probably encouraged by each of the three trade shows he or she is attending to download and install a venue app to make the participation experience better. The lifetime of the application is that of a few days, and the experience and interfaces of each of the applications are very different.

We can further assume that the traveller needs to use a ride-sharing or taxi company in each of the different cities. Many ride-sharing and taxi operators provide their own proprietary mobile application to their customers. Some of the ride-sharing operators work in many cities, whereas others (e.g. local taxi companies) operate in a limited area only. The need to install a new application for taxi services in every city is burdensome.

In one of the international cities, the traveller wishes to see a game. Obtaining the tickets requires the installation of yet another mobile application.

Even without considerations associated with travel, a consumer needs a mobile application for every bank, credit card provider, retailer, cellular service provider, and virtually any other businesses he or she interacts with. The niggling need to install and be acquainted with so many single-use mobile apps acts as an adoption barrier to the user.

The need to create, maintain and provide a mobile application is a significant technical burden on businesses as well. In some cases, the mobile app may provide a unique competitive business advantage, but in a large variety of cases, it does not. The support, development and maintenance of the mobile app on the multitude of mobile/wearable devices imposes a significant IT burden.

Instead of an abundance of mobile apps, we envision a future where there is a single application that enables the user to type in natural text (sometimes in conjunction with an image from the phone camera or a local file) to interact with a chat-server for each app. The single application can talk to the business service of any business. This business service will be supported by means of an AI model, and it would be appropriate to refer to this business service as an agent for the business. In some cases, the conversation may happen with the user providing his/her identity to the business-agent, while in other cases, the conversation may happen with the user in an anonymous mode.

As an example, the traveling business-person would be able to use this single application to find the schedule of the local public transit regardless of the city. An agent operated by the transit authority of each city allows the traveller to check the schedule and purchase tickets for the ride. An agent operated by each conference organizer lets its attendees ask questions and get directions during the operation of the conference.

5 Need for a new protocol

The vision of a common application interacting with an agent has many parallels to the browser accessing websites. The single browser application can interact with any web-site and allows the users to obtain information in either an anonymous or a signed-in/authorized model.

Despite the existence of the browser, many businesses feel the need to provide a mobile application to their users. The reason is that each company needs to extend the capabilities provided by the browser. A bank needs to provide the capabilities to check balances, transfer money, pay bills and similar operations to its clients. A local transit company needs to provide capabilities to check for schedules or purchases tickets. In the current browser design, these require providing new extensions to the markup tags provided by the underlying specifications, and any such additional tags, or conventions for transferring information requires a change on both the client side and on the server side. In effect, every bank introduces a new application layer banking protocol, each local transit company introduces a new application layer transit protocol, each airline introduces a new application layer airline protocol etc. The underlying hypertext markup language and protocol do not allow the introduction of such customization without modifying both the client and server sides.

In contrast, the emergence of large language models allows the development of an environment where a new protocol can be introduced merely by changing the capabilities on the agent side, without requiring new capabilities on the client-side. The basic function supported by the common application is the ability to converse in natural language to any server. The natural language implementation can support a generic protocol for customization by a specific agent. A basic set of conventions are required to enable characteristics such as security, authentication, improved performance, support of authorized model and the anonymous mode, etc. A large language model on the server side can provide customized protocols without requiring a new client, or any new capability on the client side.

The flexibility offered by the natural language text allows new dynamic relationships among agents to be established. As an example, suppose a company decides to partner with another company, and offer a new set of services that were not previously available in the mobile app, e.g. a conference is allowing rides from a local taxi provider with a discount. If the conference app did not have a provision to offer taxis at discount built in, a new customized app would need to be developed, or the current app modified. On the other hand, if the interactions are based on natural language and agents, a modified parsing of the text messages on the server

side would enable the addition of the new capability of discounted taxi rides for the conference. The ability to provide extensions with only server side capability augmentation can provide tremendous flexibility, which is not possible with current set of protocols.

6 NLIP scope

The scope for use of NLIP is shown in two halves of Figure 1.

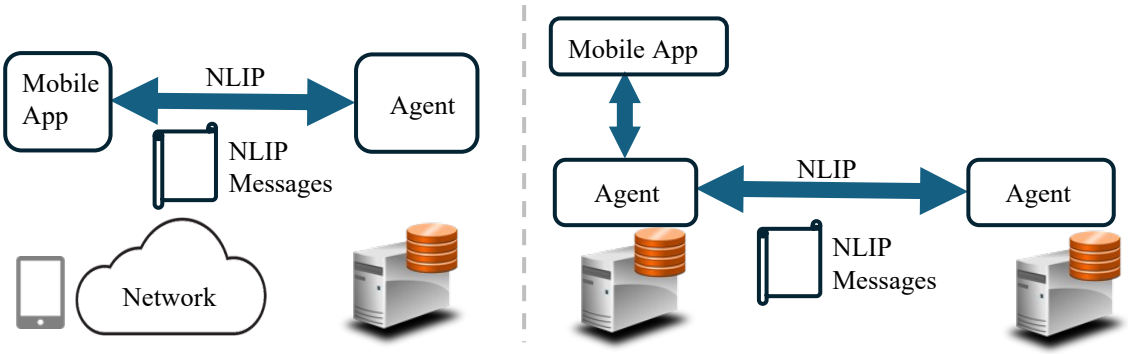

Figure 1

Figure 1 — Scope of NLIP

As shown in the left half of Figure 1, the architecture envisions a common application that provides an exchange happening using a standardized format using a standardized protocol between a mobile application and a chat service. The use of a standard convention would allow the usage of the same application across a variety of service providers. A bank, a retailer, a network service provider, an investment company -- and essentially any other businesses planning to export a chat interface to their users, would be able to use the same application to communicate without worrying about the need to provide their own chat-application.

The NLIP based component on a phone/pervasive end-point may be a stand-alone application on a mobile device, or it may be a component that gets incorporated within other mobile apps. The mobile app that incorporates NLIP may also have an embedded large language model, or it may use a human (its user) as the intelligent agent needed to interpret various modalities of information. In the former case, the mobile application may use the embedded large language model to interpret the responses from the server. In other cases, the mobile application may rely on another service running a large language model to do the interpretation.

In addition to communication between mobile apps and an Internet based service, NLIP may also be used in B2B context as communication between two businesses agents as shown in the right hand of Figure 1. In this context, the software on both ends is running on traditional servers.

The interaction shown on the right can also represent an application running on the personal machine of a user. In this case, a user may be able to use a large language model on their machine to process and interact with the responses generated by the businesses. As an example, a user AI engine may want to filter out some of the content provided by the chat application server, which are deemed to be not interesting, not useful or potentially harmful to the user on the local machine. This flexibility to the end user, that they can control information flowing into their own machine, is something which is difficult with current protocols. However, a gen-AI module under control of the user can provide the ability to manage these types of operations relatively easily.

NLIP is not designed to provide interoperability among existing business services and systems. As a migration path towards the adoption of NLIP, we envision several implementations to happen using the interaction diagram shown in Figure 2.

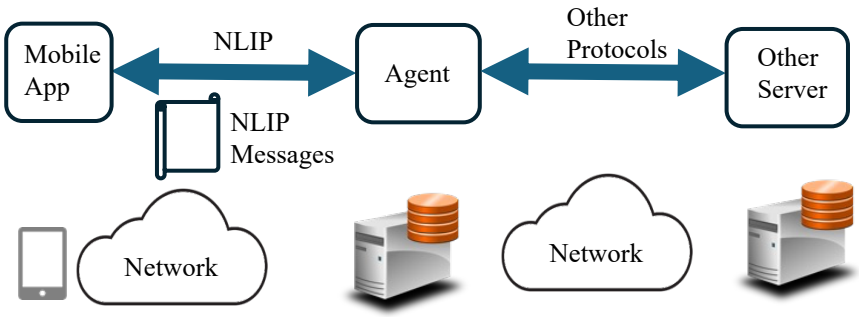

Figure 2

Figure 2 — NLIP as front-end for other protocols

In these interactions, NLIP provides a mechanism for an agent to interact with its user. However, it is not designed to act as a mechanism for interaction between an agent and other existing servers.

One advantage of adding NLIP as a interaction protocol above other protocols (e.g. an existing REST based API) is that it provides an agentic interface on existing services and functions. NLIP is designed to promote interoperability among services, and having NLIP as a common interface enables existing services to be dynamically composed into new services in a more flexible and easy manner.

Eventually, as NLIP gets adopted widely, we would envision it to be adopted by various business services as the primary interface. In this case, we can also envision NLIP proxy agents coming up which provide for interaction and communication among other agents. This interaction configuration is shown in Figure 3.

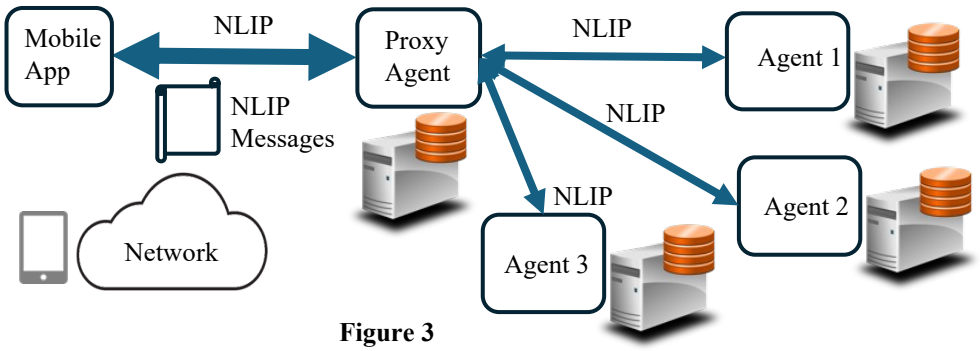

Figure 3

Figure 3 — NLIP as interaction mechanism among various agents

7 Requirements of NLIP

In order to satisfy its goal as a communication protocol between intelligent agents belonging to multiple organizations, NLIP needs to satisfy several requirements. These requirements include security, performance, and manageability requirements.

In many cases, the requirements may be satisfied by the protocols which are used for NLIP implementation. We are enumerating all requirements regardless of the layer of the protocol which implements it.

7.1 Security requirements

A key requirement for wide deployment of any protocol is the support for proper security mechanism. Towards a goal of secure communication between the clients and the servers, we need to ensure that NLIP supports

proper mechanisms for various capabilities including authentication, authorization, encryption, identity management, policy exchange support, support for anonymous mode and prevention of denial of service attacks.

Some of the security requirements are shared with performance requirements and they are described in the performance requirements in 8.3.

7.1.1 Authentication

Both interacting parties must be able to authenticate each other. NLIP must ensure authentication support using all commonly used authentication mechanisms.

7.1.2 Authorization

The authenticated remote party may only be authorized to request a restricted set of services to the local party. Appropriate mechanisms to restrict the unauthorized requests among interacting agents must be supported.

7.1.3 Encryption

The support for both current and future secure encrypted mechanisms for transfer of information must be supported. In many cases, this support may be obtained by means of underlying protocols and their support of encrypted communications.

7.1.4 Identity management

Most business and social contexts require some form of digital identity for the users to interact with them.

The notion of digital identity is quite complex. For the present discussion, we consider the simplest form of digital identity, i.e., the user login name, which uniquely identifies a user in each name space. Authentication and access control are typically performed against login names, even though today with multi-factor authentications additional information about users are leveraged for authentication. There are various approaches according to which these digital identities are issued. In the simplest form, service providers issue digital identities, with the associated credentials for authentications, to its own customers. In other cases, there are parties, referred to as identity providers, that give users digital identities, with the associated credentials. A service provider can then demand the identity management and related authentication to the identity providers. A drawback of this approach is that the identity providers end up being involved in the transactions users perform with the service providers (unless more complex protocols are used). In our context, it is important to observe that users may have different digital identities, and it is important for the NLIP being developed to understand, based on the user preferences/behaviours and the requirements of the service providers, which identity to use, or whether to even to let the user be anonymous.

7.1.5 Privacy and policy support

When clients and servers support an interactive chat, issues arise regarding the ownership and privacy of the data exchanged between the client and server. To a large extent, privacy and regulatory compliance have been relegated as an after-thought to network specifications and generative AI technologies. In order to become useful and acceptable to a broad audience, NLIP must enable a mechanism for the interacting parties to learn and be aware of each other's policies for matters such as data privacy, regulatory compliance and attribute disclosure.

7.1.5.1 Data privacy policies

Each party involved in the protocol should be able to query their peer’s policies and specify their policies for preserving the privacy of the data being exchanged between them, and mutually agree upon them prior to actual communication.

7.1.5.2 Regulatory compliance policies

Each party involved in the protocol should be able to query their peer's jurisdiction to which they comply to, and the expectations of the jurisdiction to which the other party belongs to. Some agents may refuse to communicate with parties in other jurisdictions, and some of the parties would only communicate if the other party agrees to specific terms for arbitration and conflict resolution.

7.1.5.3 Attribute disclosure policies

During a protocol interaction, each party may or may not be willing to disclosure some of their attributes, e.g. their physical location, to the other party. NLIP should enable an exchange among the parties so that they are all agreeable to the attributes being disclosed or not disclosed. A taxi server agent, for example, may insist that the clients hailing it disclose their exact physical location.

7.1.6 Anonymous mode

In many business contexts, it is important that a user be able to access services without logging in. People want to be able to access some types of information without logging in first, and businesses would like to offer some information to anyone without validating their identity. Note that some information may still need authorization and authentication of user. Examples of information that may be offered in anonymous mode include a list of items and their prices available from a retailer, the sets of flights and tickets from an airline company, the agenda of a conference etc.

7.2 Flexibility

NLIP needs to be designed to be flexible and support a multiplicity of choices for many different aspects of communication. These multiplicity of choices should include:

7.2.1 Support of multiple modalities

While text is the most common modality for user interaction using LLMs, common usage requires other modalities in the interactions as well. A bank check deposit would need images to be uploaded for the deposit, and some interactions may have the bank sending documents such as tax forms or account statements to the users. Users with accessibility needs may require use of video-based interactions. NLIP must support the various modalities required for enabling a diverse set of users and applications. Sentiment analysis based on tonal and facial analysis would be one such example of applications enabled by multi-modalities. Content encoding aside, multi-modality support has implications such as prioritizing standard interface definition for WebRTC, for mobile video chat, and additional real-time streaming performance requirements.

7.2.2 Support of multiple concurrent sessions

NLIP should enable the beginning and termination of multiple sessions, in other words, parallel conversations, between the client and the server. One session could be in the "foreground", to be the primary interacting agent interfacing with the user while others could be in the "background". An assembly of interrelated sessions forms a conversation.

A session may or may not have accompanying windows like in a browser, and each sessions will likely need to have identifiers to enable end-points to distinguish among them. Further, end-points may need to support a certain amount of information about the session in a "conversation state" such as the language(s) being used, timestamps of when the first and last utterance were transmitted, parties to the chat (possibly more than two) and so on. Since large language models can take advantage of voluminous prompts, a conversation can be set up to allow the entire history of the conversation, including timestamps and the identity (to the extent known) of the party making each utterance, to be retrievable or to limit the history kept to a fixed horizon. In some cases, end-points may want to trade-off bandwidth versus local storage to optimize their performance. NLIP should enable these interactions.

Another requirement of NLIP would be to enable headless sessions. These are agent-initiated sessions that do not directly involve a human user. As an example, an agent can query another agent for additional information in support of a user query.

7.2.3 Support of multiple underlying protocols

The dominant paradigm for applications to communicate with each other today is using some variant of REST and standardization in those cases would mean identifying the URI and parameter encoding for NLIP. At the same time, there are several other protocols that can provide alternative approaches, including but not limited to QUIC, gRPC, WebRTC, websockets and ZeroMQ. The binding of NLIP to the different underlying protocols should be defined and the specifications made in a manner so that any desired protocol binding can be supported. The main advantage of NLIP being ``hot-extensibility'' of deployed instances and continuous customization of installed interactions.

We will initially define the protocol on top of HTTPS and JSON, to take advantage of their ubiquitous availability and accessibility. However, this should not preclude the use of other underlying protocols.

7.2.4 Support of multiple platforms

Since we envision NLIP to be used across many devices and many systems, it is important that NLIP does not make implicit assumptions about programming languages and underlying hardware. The common application should be easily implementable on multiple mobile platforms, each of which has its own preferred operating system and development languages. Similarly, server-side systems may be implemented on Linux, Windows Servers, etc. using languages such as C or C++, Go, Java, JavaScript, Python, Rust, Zig, etc. NLIP specifications should support all platforms and all languages.

7.3 Performance requirements

In order to be used effectively, NLIP must support proper mechanisms for low-latency high-performance interactions. This requires supporting the following mechanisms.

7.3.1 Caching of context

Chat interactions require maintaining the context of interactions, namely the sequence of interactions that may already have happened. Instead of exchanging the context on the wire repeatedly, context should be able to be stored and reused on each side based on prior interactions.

7.3.2 Rate controls

When implementing caching mechanisms for performance, care has to be taken to allow one or both parties to specify limits on how much context they would store, and prevent denial of service attacks. The servers may want to impose limits on the length of context, or the length of chat messages they would support, or the rate at which requests may be sent to the server from a client, and vice-versa. These limits should be negotiable, and designed to enable most interactions to happen in a performant and secure manner.

7.3.3 Denial of service prevention

A common problem with any performance optimization such as saving context on both client/server side is the potential of an untrusted client or server to launch denial of service attacks, e.g. launching attacks parallel to those enabled by TCP session establishment mechanisms~\cite{harris1999tcp} such as SYN floods or half-open session state attacks. NLIP specification must take into account such potential attacks when defining any performance optimization and incorporate safeguards.

7.3.4 Real-time streaming

Interactive multi-modal chat may require separating the session into separate streams with diverse inter-stream priority and delay bound requirements. Sustained two-way communications may require support for streaming services with customized intra-stream timing and ordering requirements.

8 Protocol design using generative AI

The philosophy behind the design of NLIP is explained in this section. NLIP relies on the ability of current large AI models to be able to do a variety of translation tasks with a high degree of accuracy. In particular, NLIP relies on the ability of the large language models to translate between unstructured natural language and a structured representation such as JSON or the internal structure of a programming language effectively.

Contemporary application level protocols are developed with the assumption that all end-points of a communication and the structure being transmitted on the wire are identical. Each end-point needs to maintain the same representation as the one that is being exchanged on the wire. However, with generative AI, one can relax that requirement. On the wire, only a natural text representation can be transmitted, and each end-point can maintain its own internal representation for the structure being maintained for its tasks.

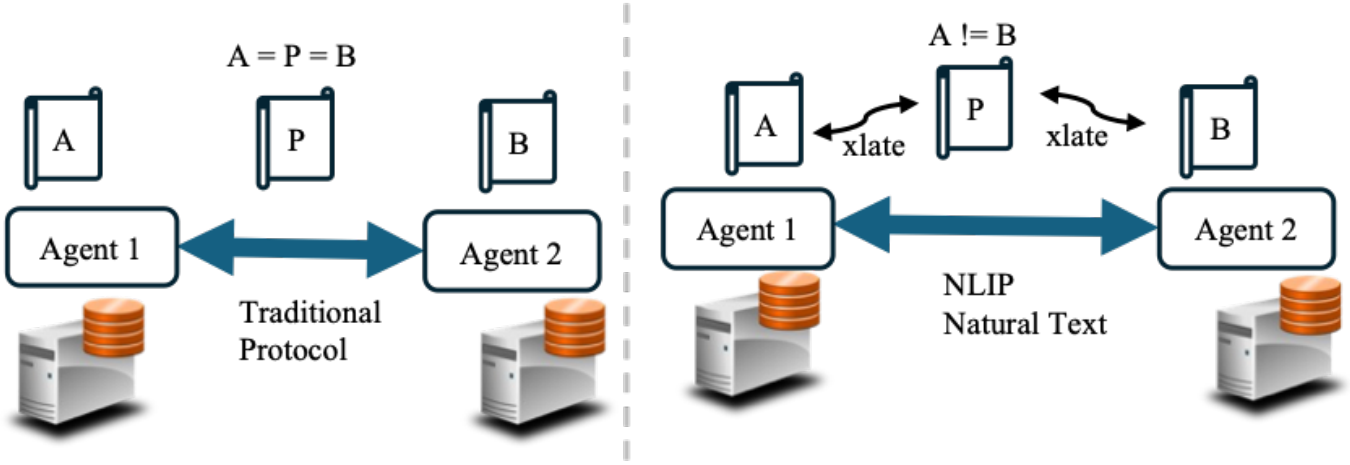

Figure 4 — Difference between NLIP and Traditional Protocols

This aspect of NLIP is shown in Figure 4. The left hand-side shows the typical design of a traditional protocol. The prevailing practice is to define a structure on the wire which the same as the structure maintained by all communicating end-points. The structure A on one end-point, structure B on the second end-point and the structure on the wire P that is carried by the protocol are all the same.

NLIP has chosen a different design point. It assumes that the structure on the wire has the same semantics as the structures on the end-points, but the format is natural language. A LLM at each end point can translate the wire format P into the local structure format A or B, as the case may be. This allows for the internal representation of structures at end-points to be independent.

As an example, let us consider the simple but illustrative case where one of the end-points is an agent that is sending its communication end-point information to the other end-point which is a registrar providing registration services for agents. The registrar maintains information about each end-point as a URL, while the agent maintains its internal information as a local host and port. The agent can register itself to the registrar on the NLIP protocol using the plain text - “I am running on host <hostname> on protocol <p>” or an equivalent natural language text representation. The registrar can use a local LLM to translate this text to an internal URL representation.

This feature of NLIP provides two key advantages. First of all, it provides resilience against upgrades of the end-points. Suppose the registrar is upgraded to a new version in which it supports each agent information as a structure consisting of a port, a protocol and a host address instead of the URL. This upgrade can be made without any impact to any of the agents. On the other hand, if the traditional approach of defining a fixed structure on the wire were to be used, the upgrade of the registrar would require changes in each of the agents interacting with it.

The second advantage is the simplification in the versioning of protocols. During any definition of a standard protocol, some key features or functions may be missed due to a variety of reasons. If a new protocol version is defined, the protocol needs to be defined to support a version number and a careful handling of mismatches in the structural differences across versions. By breaking the linkage between the internal representation of end-point structures and the wire structure, NLIP simplifies version management significantly.

The separation of the wire format from the internal structures can be viewed as a modern take on the concept of ntohs and htons – concepts developed during the early stages of computer communications development to promote interoperability between computer systems. When big-endian machines needed to talk to little-endian machines, these two macros translated between network and host structure formats. NLIP is providing the same functionality at the application level, leveraging the ability of LLMs to translate between unstructured natural text and structured representations.

It is recommended that two communicating parties exchange information about their internal representation with each other. When an agent registers with the registrar in the above example, the registrar can send a natural text representation of the internal information to the agent to validate that it has translated the text correctly to an internal representation and then translated it back. The agent can do its own internal translation and validate that the information is consistent with its internal representation. The same exchange also helps to validate if a field is missing or not incorporated properly.

9 NLIP and vendor specific protocols

In addition to NLIP, there may be proprietary vendor-specific protocols for communication among agents that may be used in specific deployments of agents. It is expected that multiple protocols may arise due to the agent development frameworks and software for end-point design provided by different vendors.

The NLIP protocol can be used in conjunction with vendor specific protocols in one of the two ways:

- Mode A: NLIP could provide the North-bound API between an agent and its client, while the agent uses a vendor specific protocol to interact with other agents or software services needed to perform its functions. It allows any NLIP client to interact with any agent using vendor-specific South-bound API.

- Mode B: NLIP could be used as the interoperability protocol that allows agents using different vendor-specific protocols to interact with each other.

These two modes for NLIP to support vendor-specific protocols are shown in Figure 5.

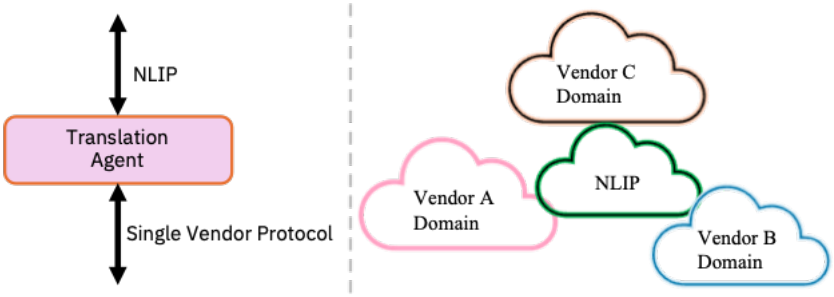

Figure 5 — NLIP and its interaction with Vendor Specific Protocols

10 NLIP use cases

In this section, we consider various use-cases of NLIP, illustrating the various NLIP messages that will be used in the exchange.

10.1 NLIP for Chat Interaction

The most common case for NLIP will be for a chat client to use it to interact with a chat server, as shown in Figure 1. In these, cases, the chat client enquiring about the nature of Ecma organization may send a message to the NLIP Server of the following nature:

```
{
     "format":"text",
     "subformat":"english",
     "content":"What is Ecma?"
}
```

The response from the chat Server will use a similar analogous format which may look like:

```
{
     "format":"text",
     "subformat":"english",
     "content":"Ecma International (Ecma) is an independent, non-profit, global
standards organization. It develops and publishes international standards for
information technology, including specifications for programming languages, data
formats, and software licensing."
}
```

Note that this simple exchange does not include the option MessageType field in its interactions.

10.2 NLIP for federation among various agents

In many cases, NLIP may be used to interact with a front-end agent which may be working with several other backend agents in order to perform a task. One such configuration of agents is the setup where a single request is sent to many different agents in the background, and the responses from those agents is collectively taken to prepare a response to the client.

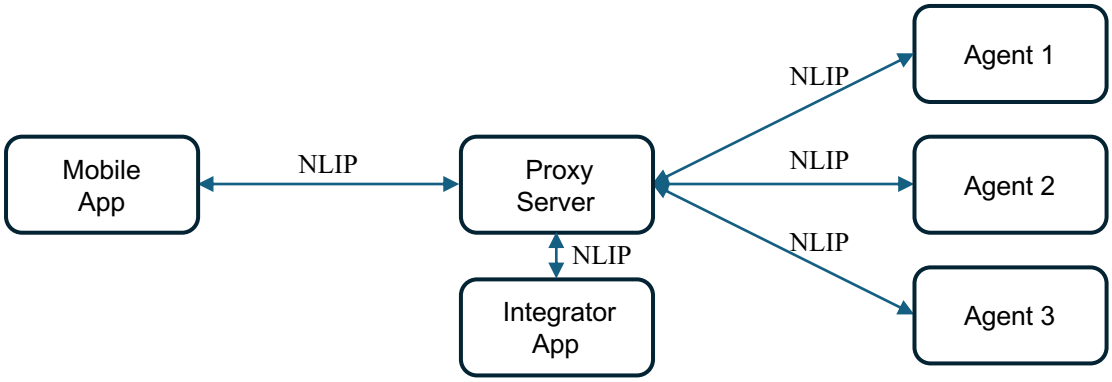

Figure 6 — Configuration of NLIP used for federation

In this system, the setup is one shown in Figure 6. A NLIP proxy server acts as the front-end to three NLIP agents and it relies on an integrator app to combine the responses obtained from individual agents.

The client can send a NLIP message to the proxy agent which may look like:

```
{
     "format":"text",
     "subformat":"english",
     "content":"What is Ecma?"
}
```

This same message is sent out to three NLIP agents by the proxy agent. The agents may reply with their own version of the information which may look like the following for different agents:

One of the NLIP servers may respond as:

```
{
     "format":"text",
     "subformat":"english",
     "content":"Ecma International (Ecma) is an independent, non-profit, global
standards organization. It develops and publishes international standards for
information technology, including specifications for programming languages, data
formats, and software licensing."
}
```

Another one may respond as:

```
{
     "format":"text",
     "subformat":"english",
     "content":"ECMA (European Computer Manufacturers Association) International
is a non-profit organization that develops and publishes standards for the
information technology (IT) industry. ECMA is known for its role in
standardizing various technologies, including programming languages, data
formats, and communication protocols.”
}
```

A third one may respond as:

```
{
     "format":"text",
     "subformat":"english",
     "content":" ECMA, or European Computer Manufacturers Association, is an
organization that was founded in 1962 to represent the interests of ICT
```

```
(Information and Communication Technology) industries in Europe. However, it is
more commonly known for its standardization work related to scripting
languages, specifically JavaScript.”
}
```

The proxy agent now needs to combine the responses from all the agents. It may send a request to an integrator agent using a NLIP messages with multiple submessages.

```
{
    "messagetype":"Request",
    "format":"text",
    "subformat":"english",
    "content":"Please combine the requests in the submessages”,
    "submessages":[
        {
            "format":"text",
            "subformat":"English",
            "content”: “Ecma International (Ecma) is an independent, non-profit,
        global standards organization. It develops and publishes international
        standards for information technology, including specifications for
        programming languages, data formats, and software licensing."
      },
      {
            "format":"text",
            "subformat":"English",
           "content":"ECMA (European Computer Manufacturers Association)
International is a non-profit organization that develops and publishes standards
for the information technology (IT) industry. ECMA is known for its role in
standardizing various technologies, including programming languages, data
formats, and communication protocols.”
        },
        {"format":"text",
        "subformat":"English",
"content":"ECMA, or European Computer Manufacturers Association, is an
organization that was founded in 1962 to represent the interests of ICT
(Information and Communication Technology) industries in Europe. However, it is
more commonly known for its standardization work related to scripting
languages, specifically JavaScript.”
        }
        ]}
```

The above is an example of a NLIP message with different submessages. The RequestType field is a private value that can be used by the proxy agent.

The response obtained from the integrator agent can be sent onward to the client by the proxy agent.

10.3 Customer support

The clause showcases how the Natural Language Interaction Protocol (NLIP) can facilitate seamless speech-to-text communication in a customer support scenario.

In this use case, a customer initiates a support request by speaking directly with a support agent using natural speech. The spoken audio is transcribed into text using a speech to text model (e.g. the NVIDIA Automatic Speech Recognition (ASR) model) .Communication between the speech processing and support agents follows the NLIP message format. Based on keywords extracted from the customer’s request, the system identifies relevant Reddit channels. A specialized search agent then queries these channels and retrieves the most recent and relevant posts, delivering them back to the customer in real time.

The speech-to-text agent transcribes spoken customer input into text. This transcribed content is then processed by a channel recommender agent which may use a model such as Granite-3.3-2B-Instruct model,

enabling contextual understanding and intent extraction. All interactions between system components adhere to the NLIP message format, ensuring modular and interoperable agent communication. Additionally, the search agent integrates MCP with Reddit's search APIs to retrieve relevant, real-time information, illustrating how multi-agent systems can dynamically support customer queries across external knowledge sources.

In this demonstration, three specialized agents—Speech to Text Agent, Channel Recommender Agent, and Search Agent—collaborate using NLIP to fulfill a customer's voice-based support request in real time. Each agent plays a distinct role in the end-to-end interaction pipeline, showcasing how modular AI components can be composed using standardized communication and interoperable APIs.

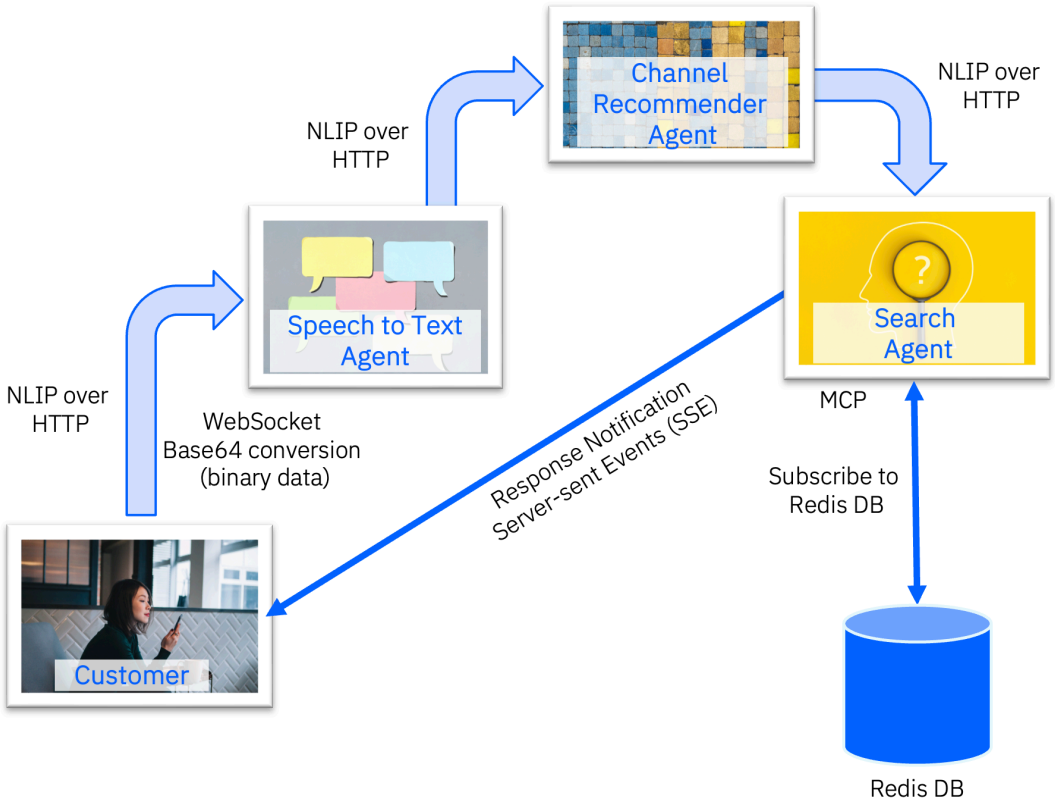

Figure 7 —

The **Speech to Text Agent** is the first point of interaction. When a customer initiates a request using their voice, the spoken input is transmitted as binary audio data—either in raw format or encoded as Base64—using a WebSocket connection. The agent receives this audio and uses a model such as NVIDIA ASR (Automatic Speech Recognition) model to convert the speech into textual form. The resulting transcribed text is then encapsulated within an NLIP message and transmitted over HTTP to downstream agents for further processing. This agent ensures that natural spoken language can be seamlessly interpreted and routed in the system using standard protocols. The NLIP message format for the Speech to Text Agent is shown in Figure 8 below.

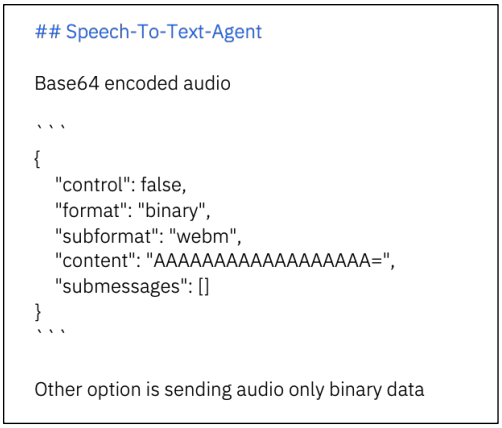

Figure 8 — NLIP message format for Speech to Text Agent

Next, the **Channel Recommender Agent** takes the NLIP-formatted transcription and analyzes the content to determine which Reddit channels are most likely to contain relevant responses. This agent performs keyword extraction or intent recognition, mapping the query to a list of *k* recommended Reddit channels. The recommended channels are then wrapped in a structured NLIP message and sent over HTTP to the Search Agent. This component plays a crucial role in narrowing the search space and improving retrieval efficiency by targeting contextually appropriate sources. The NLIP message format for the Channel Recommender Agent is shown in Figure 9 below.

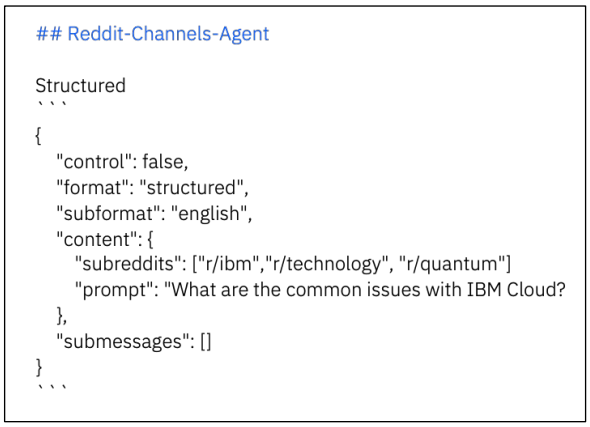

Figure 9 — NLIP message format for Reddit Channels Agent

The final component, the **Search Agent**, executes the information retrieval task. It receives the *k* recommended channels and queries, each one using the Reddit Search APIs—integrated via MCP—to fetch the top *m* most relevant or recent posts. The agent writes these results to a Redis database, publishing the response as a message. It also subscribes to Redis channels to listen for updates or follow-ups, enabling asynchronous interactions and dynamic updates. Ultimately, a curated list of *n* posts (where $n \leq k \times m$) is returned to the customer, completing the cycle. This modular architecture enables scalable, real-time information delivery by composing AI models and external data sources via well-defined agent roles and message formats. The NLIP message format for the Search Agent is shown in Figure 10 below.

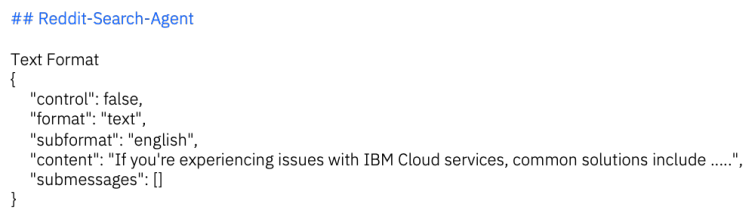

Figure 10 — NLIP message format for Reddit Search Agent

10.4 NLIP as a front end for other protocol

The NLIP server can function as a client for other protocols, allowing it to invoke remote servers corresponding to that protocol. Examples of such protocols that NLIP could as a front-end include MCP, A2A, ACP among others.

The example in 10.3 illustrates an instance of NLIP acting as such a front-end for MCP protocol.

In such systems, message exchanges adhere to the formats defined by the NLIP protocol. The NLIP server can convert them to other protocols, e.g. to the MCP protocol by incorporating the corresponding client within itself.

This use-case can be implemented in many different ways. A demonstration of this use-case is implemented in a lightweight yet powerful local environment, showcasing how multi-agent AI applications can be developed and deployed efficiently on a single client device. The entire system runs on a MacBook using Python 3.12+. The application is designed for portability and reproducibility, with configuration management handled through ***Poetry***, ensuring consistent dependency resolution across different environments.

Audio preprocessing is managed using ***ffmpeg***, which is a configurable dependency for handling a variety of audio formats and encoding schemes. This enables smooth ingestion and transformation of raw or Base64-encoded audio data into a format compatible with the NVIDIA ASR model used by the Speech to Text Agent. For dependency and script orchestration, the system employs ***uv***, a performant tool that simplifies package management and script execution within the Python ecosystem.

A ***FastAPI web server*** acts as the core of the application, providing RESTful endpoints that support the NLIP (Natural Language Interaction Protocol) message format. This allows seamless message exchange between agents, enabling modular integration and clear API boundaries. The ***NLIP SDK*** is used throughout the system to construct, parse, and manage structured messages between the speech, recommendation, and search agents. Despite its complexity, the demo runs entirely on a local MacBook, demonstrating that high-functioning, composable AI workflows can be prototyped and validated without requiring cloud-scale infrastructure.

Another important area of focus is the ***WebSocket binding for binary data***, which allows for persistent, bidirectional communication channels between client and agents. This would enable the system to more efficiently handle live audio input, maintain conversational state, and support low-latency interaction. These enhancements improve the responsiveness and robustness of the overall agentic system architecture.

Bibliography (if any)

[1] Experimental statistics, US National Bureau of Standards Handbook 91, 1963

[2] Applied Regression Analysis, Draper and Smith, Wiley Edition 2

[3] Statistical Methods for Reliability Data, Meeker, Escobar, 1998, John Wiley & Sons Inc.