Working Group
Internet-Draft
Intended status: Standards Track
Expires: 16 September 2026

Z. Chang
S. Peng
Huawei Technologies

15 March 2026

Agent Context Interaction Optimizations
draft-chang-agent-context-interaction-02

Abstract

The context distribution is important in a multi-agent system, which will impact the execution latency, token consumption, and task completion success rate, especially in a complex and multi-round workflow.

This document specifies the scenarios and procedures of agent context distribution as well as the corresponding optimization during the procedures in order to provide precise control of the context distribution among the multiple agents.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 16 September 2026.

Copyright Notice

Table of Contents

1. Introduction

In a multi-agent system, contexts such as agents' capabilities, metadata, and intermediate results are exchanged among agents through an Agent2Agent protocol to support task execution.

At present, agent context interaction commonly occurs in two scenarios: interaction among independent agents and interaction among collaborative agents. In both scenarios, context information is typically distributed in a raw or unfiltered manner, without task-aware or relevance-based processing. As a result, large volumes of redundant or irrelevant context are propagated across agents, leading to excessive token consumption.

Furthermore, existing agent interaction protocols generally lack explicit task management mechanisms, such as task state tracking and progress awareness, for complex tasks that require multi-turn interactions and the coordinated invocation of multiple agents. In the absence of structured task context, the LLM寰冭 attention may drift across interaction rounds, which can significantly increase execution latency and reduce overall task success rates.

In this document, we assume a task-oriented multi-agent control model in which a designated Master Agent is responsible for global task orchestration, while other agents are invoked on demand to execute specific subtasks. The Master Agent receives the user request, constructs the overall task, decomposes it into subtasks, and coordinates their execution by selectively invoking appropriate agents.

Invoked agents operate under the control of the Master Agent and focus exclusively on the execution of assigned subtasks. They do not maintain a global view of the task and do not directly interact with end users or other invoked agents unless explicitly mediated by the Master Agent. This master宴妬nvoked agent architecture establishes a clear separation between global task management and localized task execution.

Efficient context distribution is a critical capability in such a master宴妬nvoked multi-agent system and is largely determined by the design of the agent context interaction protocol. The protocol directly controls how much context is exchanged, how it is structured, and how it is propagated across agents. As a result, it has a direct and measurable impact on execution latency, token consumption, and task completion success rate, particularly in complex and multi-round agent workflows.

This document specifies the scenarios and procedures for agent context distribution and introduces corresponding optimization mechanisms applied during these procedures. The goal is to provide precise and task-aware control over context dissemination among multiple agents, thereby improving execution efficiency, reducing unnecessary token usage, and enhancing task completion success rates.

2. Terminology

2.1. Conventions and Terminology

TaskContext: A structured state object that is created and maintained exclusively by the Master Agent throughout the lifecycle of a complex task. It provides a persistent and machine-interpretable representation of task progress, enabling the Master agent to manage attention across multiple subtasks and prevent unintended context drift during multi-step execution.

AgentContext: The execution state that is specific to an individual agent within a multi-agent workflow. It is strictly isolated between subtasks; during each subtask invocation, only the AgentContext instance corresponding to the target agent is delivered. Upon completion, the corresponding agent returns exclusively its own updated AgentContext

SynchronousContextInteraction: A context sharing mode in which an agent MUST wait for the completion of the current interaction round, including context exchange and result feedback, before proceeding to the next task or interaction. In this mode, context distribution and task execution are strictly ordered, ensuring deterministic task progression and explicit dependency resolution among agents.

AsynchronousContextInteraction: A context sharing mode in which an agent MAY exchange contexts with multiple agents concurrently without waiting for the completion of other ongoing interactions. This mode allows parallel context distribution and task execution, enabling higher system throughput and reduced overall execution latency in multi-agent collaborative workflows.

Master Agent: The agent responsible for receiving user requests, maintaining the global TaskContext, decomposing complex tasks into subtasks, invoking other agents, and evaluating subtask execution results.

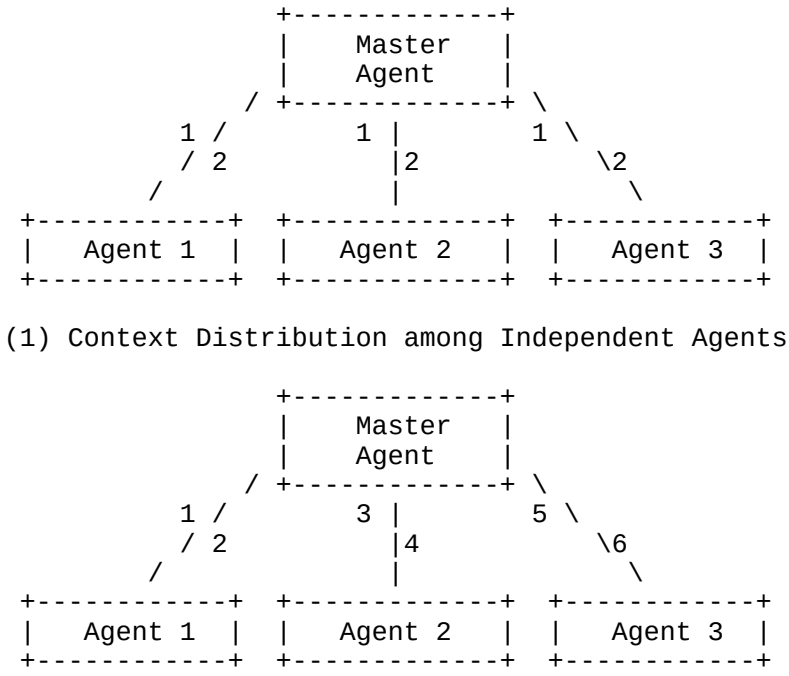
Invoked Agent: An agent that is invoked by the Master Agent to execute a specific subtask. An Invoked Agent operates on an isolated AgentContext and returns only its updated execution state to the Master Agent upon completion.

2.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Scenarios

There are two main scenarios when comes to the context distribution among agents, that is, (1) Context Distribution among Independent Agents and (2) Context Distribution among Collaborative Agents.



(1) Context Distribution among Independent Agents

(2) Context Distribution among Collaborative Agents

Figure 1: Agent Context Distribution Scenarios

In the first scenario, i.e. Context Distribution among Independent Agents, the agents in the multi-agent system are independent. The Master Agent receives the request, creates a main task and breaks down into several sub-tasks, and distribute those sub-tasks to correponding agents. The discover of the correponding agents is out of the scope of this draft. This could be for a tour-plan task, which could involve the agents of hotel-booking, flight-booking, and weather-checking. When the Master Agent receives the tour-plan task, it will break it down into sub-tasks and distribute context to each agent to accomplish the task.

In the second scenario, i.e. Context Distribution among Collaborative Agents, the agents in the multi-agent system are collaborative, that is, the input of the latter agent in a workflow will depend upon the former agent. The Master Agent receives the request, creates a main task and breaks down into several sub-tasks. It will distribute the sub-tasks to correponding agents in an order. The discover of the correponding agents is out of the scope of this draft. This could be

for a see-a-doctor task, which could involve the agents of disease-diagnosis, prescription, and buy-medicine. When the Master Agent receives the see-a-doctor task, it will break it down into sub-tasks and distribute context to the first agent, and then depending upon the output of the first agent, the second accomplishes its corresponding tasks, and then the next agent follows the same procedure.

4. Agent Context Interaction Optimizations

4.1. Overview of Agent Context Interaction Optimizations

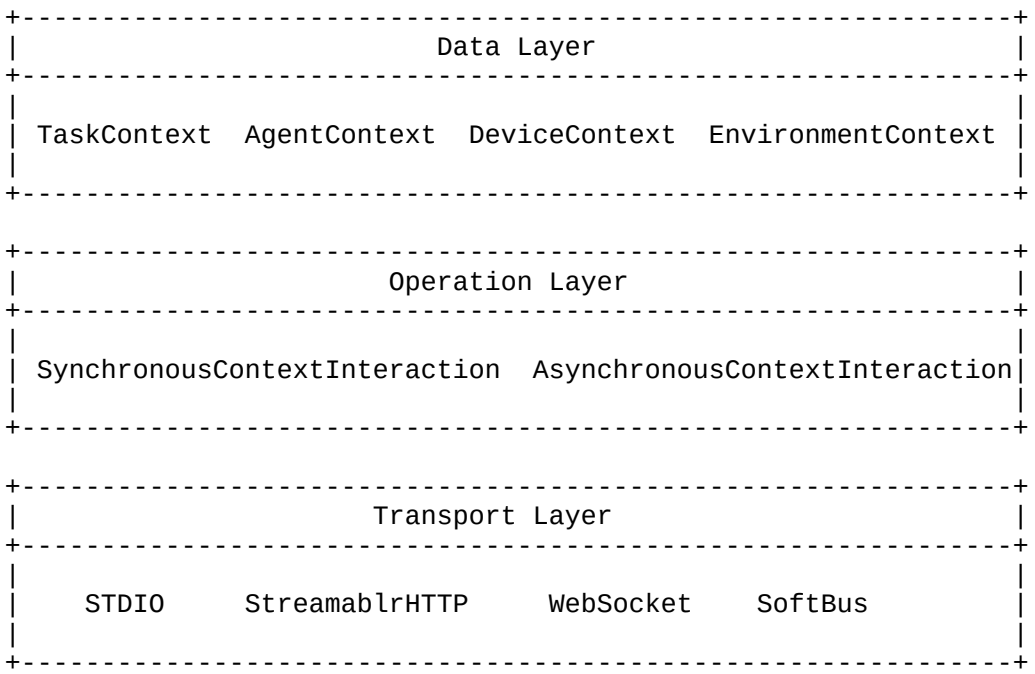


Figure 2: Architecture of the Agent Context Interaction Optimizations

We design a structured context interaction format to optimize context exchange in multi-agent systems across several key dimensions.

Agent context interaction optimizations are designed across three layers: the data layer, the operation layer, and the transport layer. At the data layer, we introduce structured context formats, including the TaskContext and AgentContext, to precisely capture and represent task and agent states. The detailed design of DeviceContext and EnvironmentContext will be done later. SynchronousContextInteraction and AsynchronousContextInteraction are desined for both scenarios mentioned in Section 3, respecitively. The corresponding transport protocols are designed to support different sceanrios.

A fine-grained task management mechanism is defined for both the Master Agent and invoked agents. Clear task progress monitoring and handling procedures are specified, which mitigate attention drift within the model and improve overall task completion success rates.

An efficient two-layer task evaluation mechanism is further established. The Master Agent evaluates the outputs of the invoked-agent based on specific fields in the structured context, avoiding the need to process or transmit full-context outputs and thereby reducing excessive token consumption.

Support for context offloading is also provided. Both the Master agent and invoked agents are allowed to store or offload full context data to external storage systems, while only structured task descriptions and references to the full context are exchanged. This approach significantly reduces the overall token consumption of the multi-agent system.

4.2. Context Isolation Strategy

A context isolation strategy is introduced to ensure that, during collaborative task execution, each agent receives only the context strictly relevant to its assigned task. Relevant context is precisely distributed to the corresponding agent in a well-structured manner, which reduces interference caused by unnecessary or unrelated contextual information.

5. JSON Specifications

5.1. TaskContext Schema

The TaskContext schema defines a structured state object that is created and maintained exclusively by the Master Agent throughout the lifecycle of a complex task. Its purpose is to provide a persistent and machine-interpretable representation of task progress, enabling the Master Agent to manage attention across multiple subtasks and prevent unintended context drift during multi-step execution.

```
 "TaskContext": {
    "TaskID": "",
    "UserQuery": "",
    "TaskName": "",
    "TaskDescription": "",
    "GoalStatus": [
        {"Goal": "", "Status": ""},
        {"Goal": "", "Status": ""},
        {"Goal": "", "Status": ""}
    ],
    "OverallStatus": "",
    "StartTime": "",
    "EndTime": "",
    "Optional": ["StartTime", "EndTime"]
    },
```

Figure 3: JSON Format - TaskContext

The TaskContext schema includes fundamental identification and intent fields such as TaskID, UserQuery, TaskName, and TaskDescription, which collectively capture the origin and semantic scope of the task. The GoalStatus array records the progress of individual goals using explicit {Goal, Status} tuples, allowing the Master Agent to track partial completion and reason over intermediate milestones. An aggregated OverallStatus field reflects the global execution state derived from these goals.

Temporal attributes StartTime and EndTime provide optional lifecycle metadata for auditing and scheduling purposes. These fields are listed in the Optional array to indicate that their inclusion is not mandatory for functional correctness and may be omitted to reduce token overhead when time information is not required.

By maintaining this structured context, the Master Agent can explicitly steer the model寰洉 focus toward active objectives, mitigate attention fragmentation across turns, and ensure consistent task execution semantics in long-running multi-agent workflows.

5.2. AgentContext Schema

The AgentContext schema represents the execution state that is specific to an individual agent within a multi-agent workflow. The messages of different agents are strictly isolated between subtasks; during each subtask invocation, only the AgentContext instance corresponding to the target agent is delivered. Likewise, upon completion, the corresponding agent returns exclusively its own

updated AgentContext to the Master Agent, without access to contexts belonging to other agents. This design enforces information isolation and prevents unintended cross-agent context contamination.

```
"AgentContext": {
        "AgentID": "",
        "AgentName": "",
        "SubTaskID": "",
        "SubTaskName": "",
        "Dependencies": [],
        "Context/ContextURI": "",
        "todoItems": [
            {"itemId": "", "description": ""},
            {"itemId": "", "description": ""},
            {"itemId": "", "description": ""}
        ],
        "ItemstateUpdates": [
            {"itemId": "", "state": 0},
            {"itemId": "", "state": 1},
            {"itemId": "", "state": 0}
        ],
        "KeyInformation": [
            {"itemId": "", "outputabstract": ""},
            {"itemId": "", "outputabstract": ""},
            {"itemId": "", "outputabstract": ""}
        ],
        "LastUpdated": "",
        "Optional": ["LastUpdated"]
    },
```

Figure 4: JSON Format - AgentContext

The AgentContext schema contains identification fields including AgentID, AgentName, SubTaskID, and SubTaskName, which bind the context to a concrete subtask instance. The Dependencies array expresses prerequisite relations among subtasks, enabling the Master Agent to reason about execution ordering and blocking conditions.

The field Context/ContextURI specifies the address of external or long-term context required for each agent to perform its task, such as retrieved documents, knowledge bases, or cached intermediate results. This indirection avoids embedding large artifacts directly in the messages and supports scalable context management.

Task execution is modelled through three coordinated structures. The todoItems array enumerates actionable steps assigned to each agent. The ItemstateUpdates array records the completion state of each item

using a binary code where 0 denotes not completed and 1 denotes completed. The KeyInformation array provides concise output abstracts generated by the agent for each item, capturing the semantic evidence necessary for result validation.

After receiving the updated AgentContext, the Master Agent evaluates subtask completion in a two-phase manner. It first inspects ItemstateUpdates for a preliminary decision; items marked as 0 are considered unfinished and bypass further assessment. For items marked as 1, the Master Agent performs a second-stage verification based on the corresponding KeyInformation, ensuring that the produced outputs satisfy the intent of the subtask.

The optional field LastUpdated records the timestamp of the most recent modification and is listed in the Optional array to indicate that its inclusion is not required for functional correctness.

5.3. Interaction Process of Agent Context Interaction Optimizations

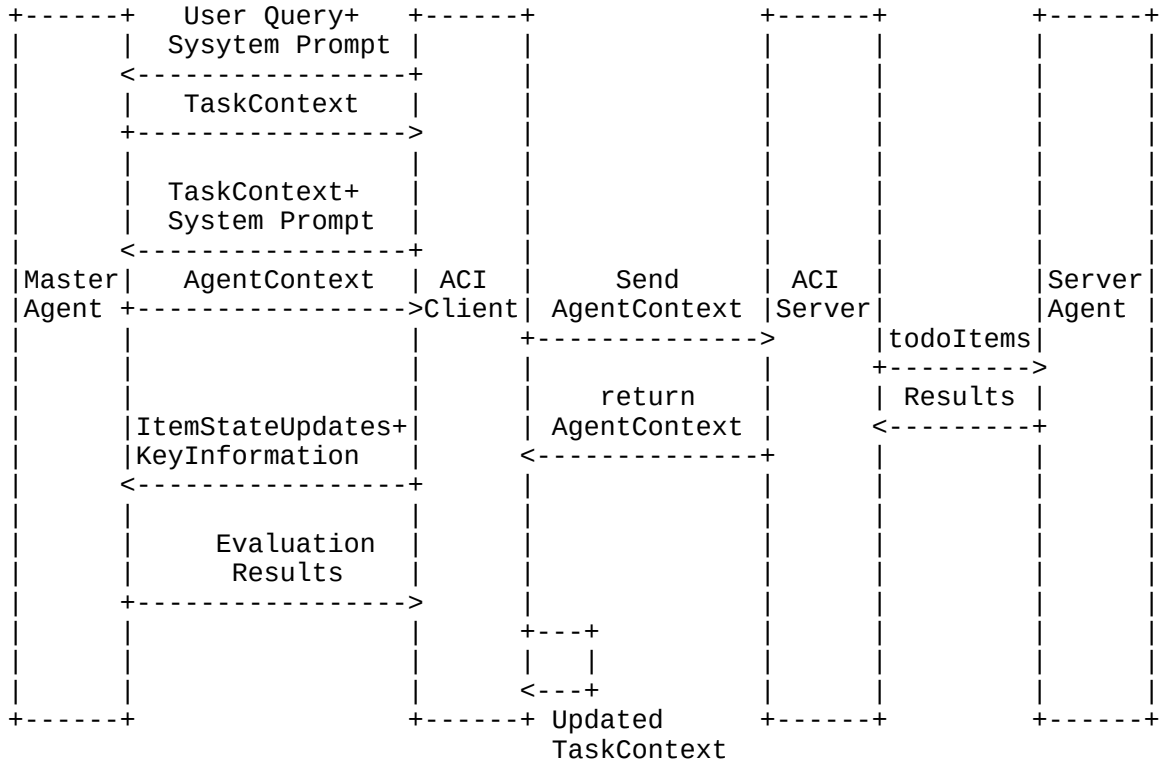


Figure 5: Interaction Flowchart of the Agent Context Interaction Optimizations

Figure 5 illustrates the interaction flow of the proposed Agent Context Interaction (ACI) optimizations in a multi-agent system. The workflow begins when the Master Agent receives a user query together with the system prompt, forming an initial TaskContext. The TaskContext is exchanged with the ACI Client, which acts as the interface for context transmission.

It is worth mentioning that the ACI Client and ACI Server are conceptual entities utilized to illustrate the logical connection between the Master Agent and the invoked agents. In practice, these components can be realized by an Agent2Agent (A2A) client and an A2A server, without introducing additional architectural constraints.

Based on the task decomposition, the Master Agent generates an isolated AgentContext corresponding to a specific sub-task and sends it through the ACI Client and ACI Server to the target Server Agent. Only task-relevant information, including todoItems and the associated context reference, is delivered to the Server Agent, ensuring precise and minimal context distribution.

After executing the assigned sub-task, the Server Agent returns an updated AgentContext containing ItemStateUpdates and KeyInformation. These updates reflect task completion status and summarized outputs rather than full execution traces. The Master Agent then evaluates the returned information to assess task progress and correctness.

Upon evaluation, the Master Agent updates the global TaskContext accordingly, enabling subsequent task scheduling or further agent interactions. This flow demonstrates how structured context isolation, incremental state updates, and layered evaluation collectively reduce token consumption, control attention drift, and improve overall execution efficiency in multi-agent collaboration.

6. Security Considerations

Agent Context Interaction Optimazation defines structured context exchange mechanisms among cooperating agents. While we do not introduce a new transport protocol, improper handling of context data may lead to security and privacy risks.

AgentContext messages may contain sensitive information, including user queries, intermediate reasoning artifacts, task metadata, and execution results. Implementations MUST ensure confidentiality and integrity of AgentContext messages by relying on secure underlying transport mechanisms, such as TLS-protected channels, and appropriate authentication and access control between agents.

The protocol promotes context isolation by design, where each agent receives only the context strictly required for its assigned task. Failure to enforce this isolation may result in unintended information disclosure or cross-task data leakage.

This document does not define agent identity formats or trust establishment procedures. Deployments are expected to rely on out-of-band mechanisms for agent authentication, authorization, and trust management.

The use of the ContextURI field introduces an indirection mechanism for accessing external or offloaded context data. While this approach reduces token consumption by avoiding the transmission of large artifacts, it also introduces additional security considerations related to pointer integrity and distributed state management.

Implementations MUST ensure the integrity and authenticity of ContextURI references. A malicious or tampered ContextURI could redirect an invoked agent to unauthorized resources or cause the retrieval of manipulated context data. Appropriate protections, such as signed references, authenticated storage endpoints, or secure capability-based access mechanisms, SHOULD be employed to ensure that only trusted agents can resolve and access the referenced context.

The lifecycle management of ContextURI references SHOULD also be carefully controlled. Storage systems that host offloaded context data SHOULD enforce explicit expiration policies (e.g., time-to-live or TTL) to prevent long-term persistence of sensitive information. Access to such storage endpoints MUST require appropriate authentication and authorization to ensure that only the intended invoked agents can retrieve the referenced context.

Deployments MUST further consider data residency and privacy implications when ContextURI references point to storage systems containing user-related information, such as UserQuery or intermediate task artifacts. Operators SHOULD ensure that storage locations comply with applicable data protection policies and that sensitive context data is protected both in transit and at rest.

The Master Agent and the storage endpoint referenced by ContextURI are required to operate within a trusted relationship. The Master Agent MUST ensure that the storage service providing context data enforces adequate security controls and maintains the integrity and availability of the stored context. Failure to ensure this trust relationship may result in context poisoning, data tampering, or unauthorized disclosure of task-related information.

7. IANA Considerations

This document does not require any immediate actions by IANA. Future extensions of the protocol may define registries or code points that require IANA assignment.

8. Acknowledgments

We would like to thank Yiyang Shao, Jinyang Li, Tao Liu, Olorundare James Kunle, Ujjwal Upadhyay, Zhuoran Ma, Muhammad Awais Jadoon and Muntaser Syed for useful discussion and ideas.

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Appendix A. Contributor Addresses

Yiyang Shao
Huawei
China

EMail: shaoyiyang@huawei.com

Tao Liu
Huawei
China

Email: liutao417@huawei.com

Jinyang Li
Huawei
China

EMail: lijinyang9@huawei.com

Figure 6

Authors' Addresses

Zeze Chang
Huawei Technologies
No. 3 Shangdi Information Rd.
Beijing
100085
China
Email: changzeze@huawei.com

Shuping Peng
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing
100095
China
Email: pengshuping@huawei.com