AGENT COMMUNICATION LANGUAGES:

STRENGTHS, WEAKNESSES, AND COMPARATIVE EVALUATION

Agent-based systems have become ubiquitous in distributed computing over the past decade or so. You see them deployed whenever autonomous entities need to coordinate toward shared objectives (Wooldridge, 2009). The central challenge? Communication. How do these agents actually talk to each other? They've got to exchange information and negotiate whilst maintaining their independence - getting this balance right is critical for the entire system to function properly.

Back in the 1990s, researchers developed Agent Communication Languages (ACLs) as a structured way to tackle this problem. These provided standardised protocols that agents could use to interact with each other (Finin et al., 1997).

This paper examines agent communication languages from both theoretical and practical angles. We've implemented a KQML/KIF-based procurement scenario to see how these formal protocols perform in practice. Alice, acting as a procurement agent, queries Bob (an inventory agent) about product specifications. This gives us something concrete to analyse rather than just discussing abstractions. The analysis doesn't limit itself to KQML/KIF though - we also evaluate alternatives including FIPA ACL, service-oriented architectures, and the LLM-based communication approaches that have started appearing recently.

. AGENT COMMUNICATION FUNDAMENTALS

The Communication Problem in Multi-Agent Systems

Multi-agent systems present communication challenges that you don't typically see in traditional distributed systems. The key difference is that agents are autonomous - they've got their own internal architectures (which can vary wildly), they might have conflicting goals, and they often don't know much about what other agents can actually do (Jennings, Sycara and Wooldridge, 1998). This creates some real headaches when they need to communicate. Getting agent communication right means dealing with three core issues. First, there's syntax - basically, how messages are structured and formatted, the way information gets encoded. Second is semantics, which is about meaning. What information is the message actually conveying? Third comes pragmatics - this is about how messages work in context, how communication helps agents achieve their goals. Agent Communication Languages try to tackle all three of these aspects at once. The aim is creating standardised frameworks so that agents built by different developers can still interact in meaningful ways.

Speech Act Theory Foundation

Most ACLs, including KQML and FIPA ACL, draw on speech act theory from philosophy of language (Searle, 1969). The basic insight from speech acts is that when we say something, we're not just conveying information - we're performing an action. Think about it: requesting something, informing someone, making a promise, issuing a command. These are all actions, not just information transfer.

This idea translates into agent systems through what are called performatives. These are speech act types that structure communication as actions. When one agent sends a message to another, it's performing an action on that agent's knowledge state (Cohen and Levesque, 1990). So instead of just pushing data around, agents are actively doing things to each other's understanding of the world.

KQML AND KIF: STRENGTHS AND WEAKNESSES

KQML Overview

Knowledge Query and Manipulation Language - or KQML as it's commonly known - came out of the ARPA Knowledge Sharing Effort back in the early 1990s (Finin et al., 1997). The basic idea was to create a message format and protocol that agents could use to communicate with each other. What's clever about KQML is how it separates different aspects of communication into layers. You've got the communication layer handling performatives and message structure, the content layer dealing with the actual knowledge being expressed (usually in KIF), and then a message layer wrapping everything up with routing and conversation management details.

Knowledge Interchange Format (KIF) serves as the content language for KQML, and it's built on first-order predicate logic (Genesereth and Fikes, 1992). What KIF does is let agents express complex propositions, queries, and assertions in a way that machines can read, regardless of how each agent internally represents knowledge. It's essentially a common language for knowledge representation.

Strengths of KQML/KIF - Semantic Clarity & Separation of Concerns

Our implementation really shows off KQML's clear semantic structure. When Alice queries Bob about 50-inch televisions, the message explicitly marks itself as an "ask-all" performative. This means she's requesting all possible answers to her query, not just one. This explicit labelling removes the kind of ambiguity you'd get with less formal approaches. Bob immediately knows what information Alice wants and what kind of response she's expecting.

The layered architecture is quite elegant when you look at it. Alice can express her procurement queries in KIF, whilst KQML takes care of the housekeeping - message routing, conversation tracking, and performative semantics. This modularity is actually pretty useful because it means you could swap out KIF for a different content language and still use the same KQML communication framework. That's real flexibility.

Conversation Management

The reply-with and in-reply-to parameters give you sophisticated conversation tracking. In our implementation, when Alice queries about HDMI specifications, her message explicitly references Bob's previous response through the in-reply-to parameter. This maintains conversational context, which becomes crucial when you need complex multi-turn dialogues. Think about negotiation or collaborative problem-solving - you need to track what's been said before.

Platform Independence

Both KQML and KIF are language and platform agnostic. You could implement the same system in Java, Lisp, or pretty much any other language, and the agents would still communicate just fine. Back in the 1990s, when integrating heterogeneous systems was genuinely difficult, this kind of interoperability was revolutionary.

Formal Semantics

Because KIF is grounded in first-order logic, it provides unambiguous meaning. Take a query like "(and (type ?product television) (size ?product 50inch))" - any KIF-compliant agent will process this the same way. There's no room for interpretation. This formal foundation means you can reason about communication, prove properties about dialogues, and verify that agents are behaving correctly.

Weaknesses of KQML/KIF - Complexity and Overhead

Building the implementation revealed just how complex KQML/KIF can get. Creating even a simple product query involves building nested KIF expressions, wrapping them in KQML performatives, and managing all the message metadata. Compare that to just calling a REST API or a direct method, and you're looking at substantial development and runtime overhead. It's a lot of work for what might be a straightforward task.

Incomplete Semantics

A problem that bothered me whilst working on this. KQML defines performatives like "ask-all" and "tell", but what do these actually mean in terms of the agents' mental states? The semantics here remain somewhat underspecified (Wooldridge, 2000). When Bob receives an "ask-all", is he obligated to respond? Does the response have to be complete? These pragmatic aspects get left to implementation, which can potentially cause coordination failures between agents.

Limited Adoption and Tooling

Despite being theoretically elegant, KQML never really took off in industry. When building our implementation, we had to construct all the KQML/KIF infrastructure from scratch - message parsing, KIF expression construction, dialogue coordination, the works. There simply aren't mature libraries and development tools available. This increases both implementation cost and the maintenance burden significantly.

Rigidity

KQML comes with a predefined set of performatives, and that can be constraining. Say Alice needs to perform some negotiation act that isn't covered by the standard performatives. Extending KQML requires careful semantic definition and getting agreement between all the agents involved. This contrasts sharply with more flexible approaches where agents can just define custom interaction patterns as needed.

Performance Concerns

The string-based representation and the complex parsing that KQML/KIF messages require introduces latency into the system. In time-critical applications or scenarios with high message volumes, this overhead might be unacceptable. Binary protocols or direct object serialisation would perform much better.

Human Readability vs Machine Processing

There's an irony here. KQML messages are theoretically human-readable, but in practice, those nested logical expressions become pretty opaque when queries get complex. Debugging multi-agent dialogues means wading through careful message logging, and the formal syntax doesn't help as much as you'd think. More structured formats like JSON or XML might actually be easier to work with for debugging purposes.

ALTERNATIVE APPROACHES TO AGENT COMMUNICATION

FIPA ACL (Agent Communication Language)

The Foundation for Intelligent Physical Agents (FIPA) developed their own ACL as an alternative that addressed some of KQML's limitations whilst keeping the speech-act foundations (FIPA, 2002). FIPA ACL differs from KQML in some important ways. On the strength side, FIPA ACL has more rigorously defined semantics - they grounded it in modal logic. They also standardised content languages with FIPA-SL rather than just relying on KIF. You get better specification of agent interaction protocols like contract net and auctions. Industry adoption has been wider too, particularly in telecommunications and e-commerce sectors. But there are weaknesses as well. FIPA ACL still shares KQML's fundamental complexity issues. Those formal semantic specifications can be difficult for practitioners to implement correctly. And there's limited support for real-time systems and streaming data. Essentially, FIPA ACL represents an evolution of the KQML approach. It addresses some semantic gaps whilst maintaining formal foundations, but you still incur the overhead of speech-act-based communication.

Direct Method Invocation and Service-Oriented Approaches

Modern distributed systems often skip formal ACLs entirely in favour of direct service invocation patterns. Think Remote Procedure Calls (RPC) or RESTful APIs. The idea is simple - rather than wrapping requests in performatives, agents just expose their capabilities as callable services. The strengths here are compelling. There's simplicity and familiarity for developers who already know these patterns. You get extensive tooling and framework support. Performance is high, especially with binary protocols like gRPC or Thrift. The semantics are clear through typed interfaces. It's become an industry-standard approach with widespread adoption. On the downside, you lose speech-act semantics - requests become just function calls. It's harder to reason formally about agent interactions. There's less support for complex dialogue patterns. Agents end up more tightly coupled through interface dependencies. And you get limited support for conversation context and multi-turn interactions. Service-oriented approaches prioritise pragmatic engineering concerns over theoretical foundations. For many real-world applications, this trade-off makes sense because you gain development velocity and system performance.

Blackboard Architectures

Blackboard systems offer a different communication paradigm altogether. Instead of agents messaging each other directly, they interact through a shared knowledge repository (Nii, 1986). The mechanism works like this: there's a central "blackboard" that stores information. Agents read from and write to this blackboard. Pattern-matching triggers agent activation when relevant information shows up. This approach has some real advantages. It decouples agents completely - they don't need to know other agents' identities. Communication patterns become flexible because any agent can respond to posted information. It's natural for opportunistic problem-solving where the solution strategy emerges dynamically. And it supports asynchronous communication naturally. But there are trade-offs. The blackboard becomes a bottleneck and a single point of failure. Implementing dialogue and conversation tracking gets difficult. Concurrency control becomes complex when multiple agents access the blackboard simultaneously. And you lose the explicit communication semantics that ACLs provide. Blackboard approaches work well for specific problem classes like signal interpretation or planning, but they lack the explicit communicative intent that ACLs offer.

Publish-Subscribe and Event-Based Communication

Event-driven architectures let agents communicate through event streams, which is quite different from the approaches we've discussed so far. The mechanism is straightforward. Agents publish events to topics. Other agents subscribe to topics they care about. Message brokers handle routing events to subscribers. The strengths are significant. Scalability is excellent because loose coupling allows independent scaling. It's a natural fit for reactive agents responding to environmental changes. Modern infrastructure like Kafka and RabbitMQ supports this pattern well. There's temporal decoupling - publishers and subscribers don't need to be active at the same time. The weaknesses matter though. Support for request-response patterns is limited. There's no built-in conversation management. You lose the semantic clarity that ACL performatives provide. Complex coordination protocols become difficult to implement. Event-based communication excels when agents are reacting to state changes rather than engaging in deliberative dialogue.

Emerging LLM-Based Agent Communication

Recent advances in Large Language Models have opened up natural language as a viable agent communication medium (Yao et al., 2023). This is quite a departure from everything we've discussed so far. The mechanism is interesting. Agents communicate using natural language prompts. LLMs interpret these messages and generate appropriate responses. Communication becomes more flexible and human-like. The strengths are compelling. It's extremely flexible - you can express virtually any communicative act. It's potentially easier for human designers to specify agent behaviours. It bridges the gap between human-agent and agent-agent communication. And you get emergent communication protocols without explicit programming. But the weaknesses are concerning. The lack of formal semantics makes verification difficult. Behaviour can be unpredictable - LLMs might misinterpret messages. There's substantial computational overhead from language model inference. Reliability becomes a concern for critical systems. And it's difficult ensuring consistent interpretation across different LLM versions. LLM-based communication represents a real paradigm shift from formal ACLs toward more fluid, adaptable interaction patterns. However, losing those formal guarantees raises serious concerns for safety-critical applications.

COMPARATIVE EVALUATION

Evaluation Criteria

To compare these agent communication approaches systematically, we need to consider multiple dimensions. How precisely is message meaning defined? That's semantic clarity. Can heterogeneous agents communicate? That's interoperability. What's the development effort and complexity? That's ease of implementation. Then there's runtime overhead and scalability for performance. The ability to express diverse communication acts covers flexibility. Can we prove properties of agent interactions? That's formal verification. And finally, there's practical adoption - industry usage and tool availability.

Comparative Analysis

Looking at the comparison table, no single approach wins across all criteria. KQML/KIF scores high on semantic clarity and formal verification but low on implementation ease and practical adoption. FIPA ACL improves on KQML's semantics but shares similar implementation challenges. RPC/REST approaches flip this - they're easy to implement and widely adopted but sacrifice semantic clarity and formal verification. Blackboard and pub-sub systems have their own trade-offs. LLM-based communication offers extreme flexibility but variable semantic clarity and very low formal verification capability.

Context-Dependent Selection

The reality is that no single approach dominates. Your optimal choice depends on what you're actually trying to build. For research prototypes and formal verification work, FIPA ACL or KQML provide the rigorous semantic foundations you need for proving system properties. For production web services, REST APIs offer simplicity, performance, and extensive tooling that outweigh formal communication semantics. If you're building event-driven systems, publish-subscribe architectures scale effectively for reactive agent systems responding to environmental changes. And for human-in-the-loop systems, LLM-based communication enables natural interaction patterns, though you sacrifice formal guarantees. Take our procurement scenario. We implemented it with KQML/KIF, but honestly, it could be done much more simply with REST APIs for a production system. However, KQML's explicit performatives would make formal analysis of the dialogue protocol easier. You could prove properties like "Alice always receives a response" or "Bob never provides information about unavailable products."

LESSONS FROM IMPLEMENTATION

Conversation Tracking Value

The reply-with and in-reply-to mechanism proved genuinely valuable for maintaining dialogue context. When Alice queries multiple products about HDMI specifications, these parameters ensure responses get correctly associated with queries. You'd have to build this yourself in simpler communication approaches.

Development Overhead Reality

Building the KQML/KIF infrastructure consumed significant development effort. We had to define performatives, construct KIF expressions, parse responses - the whole works. For what was ultimately a simple procurement query, this overhead seemed unjustified compared to just using direct database queries or REST API calls.

Debugging Complexity

Understanding what was happening between agents required careful examination of KQML message logs. The formal syntax, whilst semantically clear in theory, proved less intuitive for debugging than JSON or structured objects would have been. There's a gap between theoretical clarity and practical usability.

Limited Expressiveness in Practice

Here's something that struck me. Despite KIF's theoretical expressiveness grounded in first-order logic, our implementation used relatively simple queries. The gap between KIF's representational power and what we actually needed suggests that simpler query languages might suffice for many applications. We might be over-engineering the solution.

FUTURE DIRECTIONS

Hybrid Approaches

Combining formal ACL semantics for critical coordination with efficient binary protocols for high-volume data exchange might provide optimal trade-offs. Imagine agents using FIPA ACL for negotiation but switching to REST APIs for data retrieval. Get the best of both worlds.

Semantic Web Integration

Research into constraining and verifying LLM behaviour might enable natural language agent communication with formal guarantees. This would combine flexibility with reliability - quite an achievement if it can be pulled off.

Context-Aware Protocols

Adaptive communication protocols that adjust formality and structure based on context could balance the precision of ACLs with the efficiency of simpler approaches. The protocol adapts to what you're trying to achieve rather than forcing you into one rigid framework.

CONCLUSION

Agent Communication Languages like KQML and KIF provide rigorous frameworks for multi-agent interaction with clear semantic foundations. Our implementation demonstrated both the power and limitations of formal ACLs quite clearly. KQML enables unambiguous specification of communicative acts, and KIF provides precise knowledge representation. But the complexity and overhead of these approaches have limited their practical adoption. Alternative approaches exist - service-oriented architectures, event-based communication, and emerging LLM-based interaction. Each offers different trade-offs between formal rigour and pragmatic engineering concerns. The optimal choice really depends on your application requirements. Research systems and safety-critical applications benefit from ACLs' formal semantics, whilst production web services tend to favour simpler, more efficient communication patterns. Looking ahead, the future of agent communication probably doesn't lie in universal adoption of a single approach. Instead, we're likely to see hybrid architectures that match communication mechanisms to specific interaction patterns. As multi-agent systems become more prevalent in distributed computing, understanding the strengths and weaknesses of various communication paradigms remains essential for anyone designing these systems.

REFERENCES

Berners-Lee, T., Hendler, J. and Lassila, O. (2001) 'The Semantic Web', Scientific American, 284(5), pp. 34-43.

Cohen, P.R. and Levesque, H.J. (1990) 'Intention is Choice with Commitment', Artificial Intelligence, 42(2-3), pp. 213-261.

Finin, T., Labrou, Y. and Mayfield, J. (1997) 'KQML as an Agent Communication Language', in Bradshaw, J.M. (ed.) Software Agents. Cambridge, MA: MIT Press, pp. 291-316.

FIPA (2002) FIPA ACL Message Structure Specification. Foundation for Intelligent Physical Agents. Available at: http://www.fipa.org/specs/fipa00061/ (Accessed: 13 October 2025).

Genesereth, M.R. and Fikes, R.E. (1992) Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Stanford University.

Jennings, N.R., Sycara, K. and Wooldridge, M. (1998) 'A Roadmap of Agent Research and Development', Autonomous Agents and Multi-Agent Systems, 1(1), pp. 7-38.

Nii, H.P. (1986) 'Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures', AI Magazine, 7(2), pp. 38-53.

Searle, J.R. (1969) Speech Acts: An Essay in the Philosophy of Language. Cambridge: Cambridge University Press.

Wooldridge, M. (2000) 'Semantic Issues in the Verification of Agent Communication Languages', Autonomous Agents and Multi-Agent Systems, 3(1), pp. 9-31.

Wooldridge, M. (2009) An Introduction to MultiAgent Systems. 2nd edn. Chichester: John Wiley & Sons.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. and Cao, Y. (2023) 'ReAct: Synergizing Reasoning and Acting in Language Models', in Proceedings of the International Conference on Learning Representations (ICLR 2023).