Oops, I Heard That! Situated Communication with Locality-Aware KQML

Angelo Ferrando[1][0000−0002−8711−4670], Andrea Gatti[2][0009−0003−0992−4058], and Viviana Mascardi[2][0000−0002−2261−9926]

[1] University of Modena-Reggio Emilia, Italy, angelo.ferrando@unimore.it
[2] University of Genova, Italy, andrea.gatti@edu.unige.it, viviana.mascardi@unige.it

Abstract. In traditional BDI (Belief-Desire-Intention) Multiagent Systems (MAS), agent communication languages such as KQML (Knowledge Query and Manipulation Language) facilitate structured message exchange and are supported by widespread BDI implementations like Jason. However, KQML lacks mechanisms to account for the situatedness of agents within dynamic environments. This paper proposes an extension to KQML, KQML-S, that incorporates the notion of locality, enabling message semantics to adapt based on the agents' shared context. Specifically, we introduce a framework where agents within the same logical communication space, or "logical room", perceive indirect updates from interactions occurring within their locality. We present the theoretical foundations of KQML-S and its implementation within VEsNA (Virtual Environments via Natural language Agents), a framework extending Jason with chatbots for natural language interaction, and a Virtual Reality environment implemented in Godot.
KQML-S bridges two foundational features of agents: social ability and situatedness.
VEsNA agents, being natively situated in Virtual Reality and inheriting KQML communication from Jason, fully exploit the potential of KQML-S and showcase its usefulness in those scenarios where agents are logically or physically embodied in a discrete, "room-based" environment.

Keywords: Belief-Desire-Intention · KQML · Situated Communication · Virtual Communication Spaces · VEsNA.

1 Introduction

To achieve goals they cannot satisfy individually, agents must coordinate. This coordination typically relies on one of three mechanisms: (i) *point-to-point* private communication; (ii) *direct interaction* via coordination artifacts; and (iii) *indirect interaction* through the environment.

Examples of the first mechanism include the Knowledge Query and Manipulation Language (KQML), developed within the "Knowledge Sharing Effort" project [21], and FIPA-ACL [15], standardized by the Foundation for Intelligent Physical Agents (FIPA). Frameworks like Jason [6] and Jade [2] adopt,

respectively, KQML and FIPA-ACL. Jadescript [3] uses a subset of FIPA-ACL performatives, while Jadex [32] uses point-to-point communication but remains agnostic with respect to the language.

Coordination artifacts provide shared interaction media, giving agents access to local resources, enabling discovery of other agents, and supporting cooperative behaviors [34]. TuCSoN [31] is a well-known example, using tuple centers to support mediated coordination. In the Agents and Artifacts (A&A) meta-model [36,30], agents are proactive entities that pursue goals, while artifacts are reactive components providing shared services. This model is implemented in frameworks like CArtAgO [35] and integrated into JaCaMo [4].

Situated agents also interact indirectly with their environment, sensing and modifying it to influence others. Swarm intelligence, for example, emerges from simple agents coordinating via local interactions and environmental cues [5]. Frameworks such as NetLogo [41], Mason [28], and Repast [29] rely on the environment as the primary shared medium, with stigmergy [39,23] as the central coordination mechanism.

Point-to-point communication is typically private and lacks locality, which is instead a native feature of environment-based and artifact-mediated coordination. Locality may be physical (e.g., an agent near a pheromone trail) or logical (e.g., participation in an online chat group). However, even point-to-point messaging in Multiagent Systems (MAS) is not always private in practice.

In realistic environments – particularly those involving shared physical or virtual spaces – messages may be overheard, unintentionally propagated, or interpreted differently depending on who else is present. Traditional communication models like KQML or FIPA-ACL support direct, intentional dialogue but overlook these situated, emergent dynamics. Yet these phenomena are not peripheral: they are central to coordination and knowledge exchange in both human and artificial collectives. We formalize these dynamics – overhearing, incidental influence, and context-sensitive responsiveness – as first-class communication features. By embedding them into the communication model, we enable more context-aware reasoning and more realistic agent interaction.

KQML transport model is assumed to employ unidirectional communication links, to be reliable and order-preserving (messages are received in the order they were sent) [26]. These assumptions do not depend on the KQML language itself but on its implementation, and do not capture relevant aspects of real agent communication – see for example [11,12] for a discussion of order-preserving communication and its consequences on protocol enactability. Also, interactions shaped by spatial or social proximity are not captured by these assumptions.

Besides these limitations, KQML has also been criticized for its lack of formal semantics and the perceived arbitrariness of its performatives [37], but it is nevertheless the foundation for communication in widely adopted agent-oriented frameworks like Jason and JaCaMo, and by far a well known format for message exchange among agents: at the time of writing, the citations of [13], [26], and [14] on Google Scholar sum up to more than 4700.

Our choice to extend KQML is thus pragmatic: instead of proposing a new model, we enrich existing infrastructure with locality-aware semantics, enabling context-sensitive agent interaction without sacrificing compatibility.

Our work does not discard message routing or private messaging – in fact, KQML-S is a conservative extension of KQML and hence supports them, as long as the underlying KQML implementation does. Instead, we generalize KQML's semantics to better accommodate contexts where communication is inherently observable to co-located agents. By elevating locality to a first-class concept – rather than hiding it inside assumptions on the transport model – and distinguishing between public and private communication modes, KQML-S enables both deliberate and incidental communication in shared environments. This promotes more expressive, flexible interaction in open MAS, where transparency, coordination, and social influence are essential.

To achieve our goal we enrich communication in the AgentSpeak(L) agent-oriented programming language [33], for which Jason provides an extended interpreter [40], by incorporating locality – modeled as logical rooms populated by potentially overlapping agent subsets – into the environment configuration and by distinguishing between private and public message modes. Logical rooms are not MASs themselves: they are more similar to organizations within MASs, where agents may enter and leave, and may reside in more than one logical room at the same time. However, logical rooms may have a digital – or even physical – twin, that automatically drives how communication inside a room works. For example, if the room is physical, we may expect that agents physically located inside the room may overhear messages exchanged there, but those outside the room cannot. In our work, we also model agents' attitudes toward overheard information, influencing whether and how they respond to it. Our formal setting is AgentSpeak(L)'s speech act semantics, using KQML as the base communication language and Jason as the implementation platform. We focus on the KQML `Achieve` and `Tell` performatives. The resulting extension, KQML-S, is integrated into VEsNA [19,20,18], a framework that connects Jason agents to Virtual Reality environments developed in Godot [22], and enables natural language interaction. Because agents in VEsNA are spatially situated in Virtual Reality (VR), the adoption of KQML-S, with logical rooms corresponding to physical rooms in VR, is both natural and effective.

2 Motivation

Let us consider five scenarios taken from daily life of academic computer scientists.

Scenario 1. Alice enters the open space of her lab and says to her colleague Bob "*Hi Bob, I'm late... I forgot to make two further copies for today's exam, may you kindly print them?*". Alice believes that Bob – who is an early bird – is already at his desk, not visible from Alice's desk, and that the point to point `Achieve` message just sent was properly received. She starts collecting all the stuff she needs for running the exam, counting on the two further copies

appearing soon, but Bob is enjoying a coffee in another room and does not hear. Alice has to print the copies herself, the toner is low, the paper is missing, and the exam starts late...

Scenario 2. At the other extreme, a message is (unexpectedly) heard by too many agents. Alice enters the lab very early in the morning, when Bob is - usually - the only one already there, and she says "*Hi Bob, we won that big project on BDI agents with Carol... We have plenty of money, but do not tell the PhD students...*". Today Frank, George, and Hilary, three PhD students, have a deadline and are already in the lab, although not visible from the door. The point to point `Tell` message was meant from Alice to Bob but the space where it was sent was not a private one. What Alice says when she is in her lab, cannot be heard elsewhere, as locality poses constraints on which agents may receive messages that were not directed to them. However, Frank and George – that believe everything they hear – have already recorded that there is more money for them. Hilary is instead skeptical, and just ignores it.

Scenario 3. Let us suppose that Alice and Bob are in the same WhatsApp group with Carol: a safer way for Alice to share the good news of the funded project would be to send a WhatsApp message there. Even if Alice and Bob are in the same physical room, other "logical rooms" exist: they can be in more rooms at the same time, with different persons. Although incautious communications may lead to eavesdropping, it may also happen that unexpected hearing is beneficial. Let us consider a fourth scenario.

Scenario 4. Alice asks Bob to look for the funded projects' rules, but Bob is sipping his second coffee somewhere else. However, Dave is at his desk. He hears the request and, being altruistic, says "*I can do it*" to Alice. Although Alice asked Bob for the favor, she happily accepts Dave's help.

A last, different situation is the following.

Scenario 5. Alice enters the lab and says "*Hi, may anyone print one copy of the EMAS 2025 call for papers? It's online...*". We distinguish this message, having a generic "anyone in the lab, although I do not know who is in" recipient, from a broadcast message intentionally sent to a specific set of receivers, or to *all* the agents in the MAS. In fact, in this case, the request might be heard by Irene, a new PhD student that Alice does not even know, who is in the lab and volunteers to make the print. Bob is at the coffee machine in another room and, again, sneaks out of work. Messages sent to anyone in a logical room raise the problem of coordination among those persons that might achieve the goal, resulting into dozens copies of the paper printed. It will be up to Alice to decide to whom assigning the task.

These examples show that also KQML-like speech-act based communication, by default point-to-point and mediated neither by coordination artifacts nor by the environment, may be affected by situatedness. When the agent framework environment is implemented in Virtual Reality and consists of rooms – or regions – therein, communication must take them into account.

3 Preliminaries

In this section, mainly based on the paper by Vieira et al. [40], we introduce the foundational concepts necessary to understand the paper.

AgentSpeak(L). AgentSpeak(L) is a logic programming language that provides an abstract framework for programming BDI agents [33].
The beliefs of an agent determine what an agent currently knows about itself, the other agents in the system, and the environment. They are defined as atomic formulae, as follows:

$$b \ ::= \ P(t_1, \ldots, t_n) \quad (n \geq 0)$$

where P denotes a predicate symbol, and $t_1, \ldots, t_n$ are standard terms of first-order logic. A belief base is a sequence of beliefs:

$$\mathit{beliefs} \ ::= \ b_1 \ldots b_n \quad (n \geq 0)$$

The beliefs defined by the programmer at design time make up for the initial belief base. The rest of the beliefs are then added dynamically during the agent's lifetime.
An achievement goal in AgentSpeak(L) is specified as:

$$g \ ::= !at$$

where at is an atomic proposition.
Finally, an action in AgentSpeak(L) is defined as:

$$a \ ::= \ A(t_1, \ldots, t_n) \quad (n \geq 0)$$

Action are written using the same notation as predicates, except that an action symbol A is used instead of a predicate symbol.
Plans are used to define the course of action for the agent to fulfill its goals. A plan has three main components: a triggering event te, denoting the event triggering the execution of the plan, a context $ctxt$, denoting the conditions that must hold to consider the plan applicable, and a body h consisting of a sequence of steps to be executed. A plan in AgentSpeak(L) is defined as:

$$p \ ::= \ te : ctxt \leftarrow h$$

The triggering event is defined as follows:

$$te \ ::= \ +b \mid \ -b \mid \ +g \mid \ -g$$

meaning the addition (resp. deletion) of a belief b, and the addition (resp. deletion) of a goal g. A plan is relevant for a triggering event if the event can be unified with the plan's head.
For a plan to be considered applicable a condition $ctxt$ must hold as a logical consequence of the agent's belief.
The body of a plan is composed of actions (a), belief updates ($+b$, $-b$), and achievement goals (g). We omit test goals for brevity. The sequence of formulae denoting the body of a plan is:

$$h \ ::= \ a \mid g \mid \ +b \mid \ -b \mid h; h'$$

An agent program contains a plain library with a set of plans:

$$\mathit{plans} \ ::= \ p_1 \ldots p_n \quad (n \geq 1)$$

Finally, we define an agent through its beliefs and plans:

$$\mathit{agent} \ ::= \ \mathit{beliefs} \ \ \mathit{plans}$$

KQML. One of the earliest formal definitions of KQML semantics was provided by Labrou and Finin [25], building on the foundational work of Cohen and Perrault on action-theoretic semantics for natural language speech acts [8]. Their key insight was that if utterances are considered actions, then formal action-reasoning frameworks, such as STRIPS-style pre- and post-conditions, can be applied to model their effects. Specifically, Cohen and Perrault used this approach to define the semantics of the "inform" and "request" speech acts, framing them in terms of the beliefs, desires, and abilities of conversation participants.

Pre-condition(S):	$\text{bel}(S, X) \wedge \text{know}(S, \text{want}(R, \text{know}(R, \text{bel}(S, X))))$
Pre-condition(R):	$\text{intend}(R, \text{know}(R, \text{bel}(S, X)))$
Post-condition(S):	$\text{know}(S, \text{know}(R, \text{bel}(S, X)))$
Post-condition(R):	$\text{know}(R, \text{bel}(S, X))$
Action Completion:	$\text{know}(R, \text{bel}(S, X))$

Fig. 1. Semantics for `Tell` (Labrou & Finin, 1994).

Figure 1 illustrates the semantics of the KQML performative `Tell(S, R, X)` (where S informs R that it believes X to be true), following [25].

With `Achieve`, R is asked to want to try to make the content X true of the system. With respect to `Tell`, the `Achieve` performative makes sense when the receiver R has a representation of the real world in its belief base and the result of the attempt to "make the content true" is some action in the real world [26].

3.1 Speech Act Communication in AgentSpeak(L)

KQML messages in AgentSpeak(L) follow the format $\langle mid, id, ilf, cnt \rangle$, where mid uniquely identifies the message, id specifies the recipient when sending or the sender when receiving, ilf denotes the illocutionary force (namely, the performative), and cnt contains the message content.

Messages are exchanged asynchronously and stored in a mailbox, with one message processed at the start of each reasoning cycle. The transition system manages three key sets: M_{In}, which holds received but unprocessed messages; M_{Out}, containing messages awaiting transmission; the set of suspended intentions awaiting responses to previously sent information requests is also needed in the general case, but not in this paper.

Messages of type "ask" (`AskIf`, `AskAll`, `AskHow`) suspend intentions and we do not deal with them in this paper. The rule *ExecActSnd-noAsk* below provides the semantics of sending a message different from an ask one. To keep the presentation simple, we discuss only those elements in the configuration of an agent, that are affected by the `.send` action.

$$\text{(ExecActSnd-noAsk)}\ \frac{\text{next action to be executed by } ag \text{ is } .\text{send}(id, ilf, cnt) \quad ilf \notin AskSet}{\langle ag, C, M, T, ExecInt \rangle \rightarrow \langle ag, C', M', T, ClrInt \rangle}$$

$$\begin{array}{ll} where & AskSet = \{AskIf, AskAll, AskHow\} \\ & M'_{Out} = M_{Out} \cup \{\langle mid, id, ilf, cnt\rangle\} \\ & \qquad \text{with } mid \text{ a new message identifier;} \\ & .\text{send}(id, ilf, cnt) \text{ is removed from the intention it belonged to} \end{array}$$

The semantics of sending involve adding a well-formed message to the agent's mail outbox, as defined by the *ExecActSnd* rule. For non-suspended intentions, the cycle proceeds with *ClrInt*, ensuring that the updated intention – now without the sending action – undergoes the necessary clearing.

On the receiver side, receiving a message with `Tell` performative and Bs content, where Bs is a set of beliefs, has the following semantics:

$$\text{(Tell)}\ \frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, Tell, Bs\rangle \\ (mid, i) \notin M_{SI} \quad \text{for any intention } i \end{array}}{\begin{array}{c} \langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag', C', M', T, SelEv\rangle \\ \begin{array}{ll} where & M'_{In} = M_{In} \setminus \{\langle mid, id, Tell, Bs\rangle\} \\ & \text{for each } b \in Bs \\ & \quad ag'_{bs} = ag_{bs} + b \\ & \quad C'_E = C_E \ \cup \ \{\langle +b[id], T\rangle\} \end{array} \end{array}}$$

A `Tell` message can be received either as a reply or as an `inform` action. When received as an `inform`, the AgentSpeak(L) agent incorporates the message content into its belief base, tagging the sender as the source of that information. This reflects the "action completion" condition outlined by [25], where the receiver acknowledges the sender's perspective on the belief.

The rule for receiving a message with `Achieve` performative and at content, where at is an atom, is

$$\text{(Achieve)}\ \frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, Achieve, at\rangle \\ (mid, i) \notin M_{SI} \quad [\text{for any intention } i] \end{array}}{\begin{array}{c} \langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag, C, M', T, SelEv\rangle \\ \begin{array}{ll} where & M'_{In} = M_{In} \setminus \{\langle mid, id, Achieve, at\rangle\} \\ & C'_E = C_E \ \cup \ \{\langle +!at, T\rangle\} \end{array} \end{array}}$$

In an appropriate social context (e.g., when the sender holds authority), receiving an `Achieve` message prompts the agent to execute a plan triggered by `+!at`, attempting to fulfill the goal specified in the message. This adds an external event to the event set, linked to an empty intention (T). With `Achieve` messages, new intention stacks can be created directly from incoming goals, and these plans may themselves include further achievement goals, pushing additional plans onto the stack.

The rule *MsgExchg* provides a semantics to KQML communication at the MAS level:

$$\text{(MsgExchg)}\ \frac{\langle mid, id_j, ilf, cnt\rangle \in M_{id_i Out}}{\begin{array}{c} \{AG_{id_1}, ..., AG_{id_i}, AG_{id_j}, ..., AG_{id_n}, env\} \rightarrow \\ \{AG_{id_1}, ..., AG'_{id_i}, AG'_{id_j}, ..., AG_{id_n}, env\} \end{array}}$$

$$\begin{aligned} where \quad & M'_{id_{iOut}} = M_{id_{iOut}} \setminus \{\langle mid, id_j, ilf, cnt\rangle\} \\ & M'_{id_{jIn}} = M_{id_{jIn}} \cup \{\langle mid, id_i, ilf, cnt\rangle\} \end{aligned}$$

It means that if agent id_i sends a message to agent id_j in a MAS where there are id_1, ..., id_n agents, the only two agents affected by the communication are id_i, whose mailbox is updated by removing $\langle mid, id_j, ilf, cnt\rangle$ and id_j whose mailbox is updated by adding it. The environment env is not affected by the communication.

From a technical point of view, Jason provides a `.send( Receiver, Performative, Cnt, Answer, Timeout )` action where `Receiver` can be either the name of another agent or a list of names, `Performative` is one of the known KQML illocutionary forces, or performatives (`tell`, `askOne`, `achieve`, ...), `Cnt` is the content, `Answer` is an optional parameter used with `ask` messages to unify the answer from `Receiver` and `Timeout` is an optional parameter that sets a timeout in case the agent is waiting for an answer. When `.send` is called, the message is sent.

4 Formalization of KQML-S

In this section, we present the design of KQML-S. We begin by illustrating how the standard KQML semantics must be adapted to account for the situatedness of agents. Following this, we show how KQML-S can be integrated into the operational semantics of AgentSpeak(L). Both modifications are implemented modularly, preserving the existing syntax and semantics of KQML and AgentSpeak(L). New performatives and their corresponding semantics are introduced, ensuring full backward compatibility.

4.1 How to make KQML situated

We now focus on extending KQML to incorporate the situatedness of agents within a MAS by introducing the concept of a *room*, which serves as a logical representation of agent locality.

We identify the set of logical rooms with *Rooms*. *Rooms* may evolve over time, as does the set $Ags = \{AG_{id_1}, ..., AG_{id_i}, AG_{id_j}, ..., AG_{id_n}\}$ of agents in the MAS, and the enviroment env. For example, logical rooms might include the `coffee machine room`, the `MAS lab`, and a `WhatsApp chat`. It is important to note that a logical room does not need to correspond to a physical space in the real world; for instance, the `WhatsApp chat` represents a logical communication group rather than a physical location.

Given a set of rooms, we define a function $loc : Ags \to \mathcal{P}(Rooms)$ mapping each agent to a set of logical rooms the agent currently occupies.

The first data structure that must be extended is the one modeling the MAS: in KQML-S, we add a loc element to the tuple, that becomes

$$\{AG_{id_1}, ..., AG_{id_i}, AG_{id_j}, ..., AG_{id_n}, env, loc\}$$

All the agents in the MAS are, by default, in the *mas* logical room and they cannot exit it, unless they leave the MAS. If one agent is in more than one room, it will receive all the messages – either private and directed to it, or public – that are sent by agents in that room. The values associated by *loc* with agents may change over time, as agents move/act, and new logical rooms might become available.

The second data structure that we must change is the one for messages, that is extended with a *mode*. In the simplest setting that we address in this paper, *mode* is a couple whose first element is one room identifier (*mas* for the logical room where all the agents are located, by default), and the second element is either *public* or *private*. Independently from the room, agents may communicate privately, for example using private chats on social media, or whispering in a crowded room, or in *public* mode, meaning that communication is meant to be received by one agent, but both the sender and the receiver (should...) know that all the other agents in the room will hear what they say.

A KQML-S message looks like $\langle mid, id_j, ilf, cnt, (room, how)\rangle$. The id_j receiver can assume the *all* value. This should not be confused with a broadcast message. In broadcast, we expect that the list of receivers is known to the sender: we do not cope with broadcast explicitly, as it can be resorted to as many individual point-to-point private messages as the intended recipients. The *all* receiver identifier, instead, stands for "all agents in *room*, whoever they are". For simplicity and coherence with our intended use, *all* is allowed only in *public* mode.

The standard KQML semantics is a special case of the KQML-S one obtained by setting $loc = \{(AG_{id_1}, \{mas\}), ..., (AG_{id_i}, \{mas\}), ..., (AG_{id_n}, \{mas\})\}$ and $mode = (mas, private)$. Apart from the addition of *loc* and *mode*, only a few rules of the KQML semantics are affected by the KQML-S extension.

4.2 Extension of global rules

In KQML-S the behavior of the MAS remains the same as in KQML, if communication is *private*. However, if it is *public*, all the agents in the same room as the speaker are affected. Interestingly, the receiver of the message might not be affected, if it is not in the same room as the speaker. And interestingly, in the default setting, the speaker will still believe that the intended receiver got the message, and will make plans accordingly.

The KQML-S extension of *MsgExchg* is

$$\text{(MsgExchg-S one)}\ \frac{\langle mid, id_j, ilf, cnt, (room, public)\rangle \in M_{id_{iOut}}}{\begin{array}{c}\{AG_{id_1}, ..., AG_{id_i}, AG_{id_j}, ..., AG_{id_n}, env, loc\} \rightarrow \\ \{AG'_{id_1}, ..., AG'_{id_i}, AG'_{id_j}, ..., AG'_{id_n}, env, loc\}\end{array}}$$

$$\begin{array}{lll} where & 1. & M'_{id_{iOut}} = M_{id_{iOut}} \setminus \{\langle mid, id_j, ilf, cnt, (room, public)\rangle\} \\ & 2. & M'_{id_{jIn}} = M_{id_{jIn}} \cup \\ & & \qquad \{\langle mid, id_i, ilf, cnt, (room, public)\rangle\} \\ & & \quad \text{if } room \in loc(id_j); \end{array}$$

3. $M'_{id_{kIn}} = M_{id_{kIn}} \cup$
$\{\langle mid, id_i, Notify, sent(id_j, ilf, cnt), (room, public)\rangle\}$
for all agents $id_k \neq id_j$ such that $room \in loc(id_k)$;

4. $M'_{id_{kIn}} = M_{id_{kIn}}$
for all agents id_k (possibly including id_j) s.t. $room \notin loc(id_k)$.

The semantics is as follows: if agent id_i sends a message to id_j in *public* mode (analogous to speaking loudly in a room), all agents present in the same room will become aware that the message was sent. The intended recipient, id_j, if present in the room, will receive and process the message directly, triggering the standard message-handling mechanism (Condition 2: $M_{id_{jIn}} \cup \{\langle mid, id_i, ilf, cnt, (room, public)\rangle\}$). All other agents in the room will update their knowledge to reflect that id_i sent a message to id_j (Condition 3: $M'_{id_{kIn}} = M_{id_{kIn}} \cup \{\langle mid, id_i, Notify, sent(id_j, ilf, cnt), (room, public)\rangle\}$). Agents not present in the room – including id_j if located elsewhere – remain unaffected (Condition 4).

The `Notify` performative is new in KQML-S, and its effect on the agent's semantics is described in Section 4.3.

In KQML-S, messages sent in public mode retain their recipient field. This field defines the sender's communicative intent, which is used both by the designated receiver and by overhearing agents to interpret the social context of the message. For example, an overhearing agent may infer that the message was not intended for them, and adjust their behavior accordingly (e.g., ignoring it, intervening altruistically, or reacting with skepticism). Thus, the recipient field is essential for interpreting the semantics of overheard messages.

A different situation is when Alice says *"May anyone print this paper?"*, without directing the information to anyone. This implicitly means that it is directed to all the colleagues in the lab, and can be modelled by setting *all* as the intended receiver:

$$\text{(MsgExchg-S all)}\ \frac{\langle mid, all, ilf, cnt, (room, public)\rangle \in M_{id_{iOut}}}{\begin{array}{c}\{AG_{id_1}, ..., AG_{id_i}, AG_{id_j}, ..., AG_{id_n}, env, loc\} \rightarrow \\ \{AG'_{id_1}, ..., AG'_{id_i}, AG'_{id_j}, ..., AG'_{id_n}, env, loc\}\end{array}}$$

where
1. $M'_{id_{iOut}} = M_{id_{iOut}} \setminus \{\langle mid, all, ilf, cnt, (room, public)\rangle\}$
2. $M'_{id_{kIn}} = M_{id_{kIn}} \cup \{\langle mid, id_i, ilf, cnt, (room, public)\rangle\}$
for all agents id_k such that $room \in loc(id_k)$.

Differently from *MsgExchg-S one*, *MsgExchg-S all* changes the message queues of all the agents in the room in the same way, as if all of them were the intended receivers.

4.3 Extension of local rules

In the previous section, we examined how the operational semantics of AgentSpeak(L) must be adapted to account for the situatedness of agents involved in

the communication. This led to the introduction of a new performative, `Notify`, used to create messages that inform agents within the same logical room where the communication occurs.

In this section, we detail how AgentSpeak(L)'s semantics is extended to handle this addition. Specifically, we introduce two new rules to manage messages with the `Notify` performative. In our framework, notifications relate to either a `Tell` or an `Achieve` messages, sent to one agent sharing the same logical room, as outlined in the extended global rules. Although additional rules could be defined to cover other interpretations of `Notify`, this work focuses on these two primary and common cases to maintain clarity and conciseness.

The first rule, *NotifyTell*, handles the notification of a `Tell` message. This rule governs the reception of the notification and updates the agent's belief base based on its current *mood*. The mood set defines characteristics that influence an agent's behavior when receiving `Tell` or `Achieve` messages. An agent may be `Credulous` and/or `Altruistic`. We assume a closed world assumption in interpreting the agent's mood: if an agent is not `Credulous`, it is skeptical. If it is not `Altruistic`, it is selfish. There is no need to be explicit on being skeptical or selfish: they are just a consequence of not being `Credulous` or `Altruistic`. If the agent is in an `Credulous` mood, it fully trusts the information overheard in its logical room and updates its belief base as if it were the direct recipient of the original `Tell` message.

$$\text{(NotifyTell)}\ \frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, Notify, sent(\_, Tell, Bs), mode\rangle \\ (mid, i) \notin M_{SI} \quad \text{for any intention } i \end{array}}{\langle ag, C, M, T, ProcMsg, mood\rangle \rightarrow \langle ag', C', M', T, SelEv, mood\rangle}$$

$$\begin{array}{ll} where & M'_{In} = M_{In} \setminus \{\langle mid, id, Notify, sent(\_, Tell, Bs), mode\rangle\} \\ & \text{if } Credulous \in mood \text{ then} \\ & \quad \text{for each } b \in Bs \\ & \quad\quad ag'_{bs} = ag_{bs} + b \\ & \quad\quad C'_E = C_E \ \cup\ \{\langle +b[id], T\rangle\} \end{array}$$

The second rule, *NotifyAchieve*, handles the notification of an `Achieve` message. The agent's response to this notification depends on its current *mood*. If the agent is in an `Altruistic` mood, it checks whether it possesses any relevant plans – applicable given its current beliefs – that can achieve the goal overheard in the logical room. In this scenario, the receiving agent recognizes that the sender requires assistance in achieving a goal and acknowledges its own ability to help. However, it would be inappropriate for the agent to autonomously pursue the goal without coordinating with the sender. To address this, the *NotifyAchieve* rule specifies that if the notified agent can assist (*i.e.*, it has an applicable plan), it sends a message to the sender, offering to achieve the goal on their behalf. What follows is domain-dependent and must be programmed by the developer: it is up to the sender agent to autonomously decide whether to accept the offer, delegate the goal, or continue pursuing other strategies.

Similar to the previous rule, if the agent is not in an `Altruistic` mood, it simply discards the message and takes no action to assist in achieving the overheard goal.

$$\text{(NotifyAchieve)}\ \frac{\begin{array}{c} S_M(M_{In}) = \langle mid, id, Notify, sent(\_, Achieve, at), mode\rangle \\ (mid, i) \notin M_{SI} \quad \text{[for any intention } i\text{]} \end{array}}{\langle ag, C, M, T, ProcMsg, mood\rangle \rightarrow \langle ag, C, M', T, SelEv, mood\rangle}$$

$$\begin{array}{ll} \textit{where} & M'_{In} = M_{In} \setminus \{\langle mid, id, Notify, sent(\_, Achieve, at), mode\rangle\} \\ & \text{if } Altruistic \in mood \text{ then} \\ & \quad \text{if } AppPlans(ag_{bs}, RelPlans(ag_{ps}, at)) \neq \emptyset \\ & \quad\quad M'_{Out} = M_{Out} \cup \{\langle mid', id, Tell, can\_achieve(at), mode\rangle\} \end{array}$$

5 Implementation

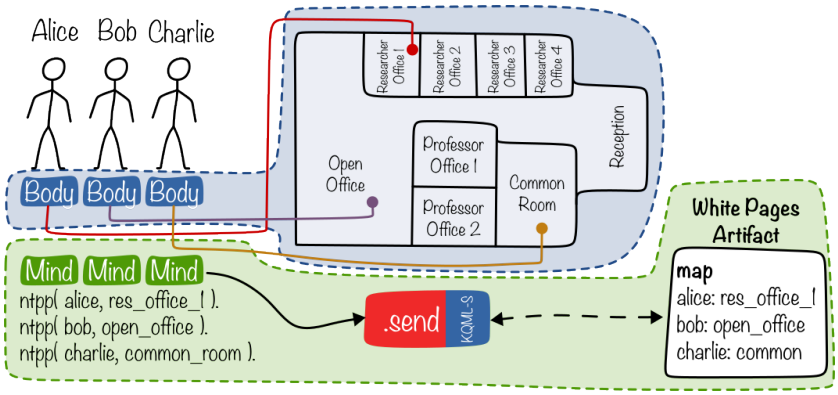

Fig. 2. VEsNA KQML-S design. Each agent has a body and a mind, it is physically located inside a region, and models (and stores) this information using Region Connection Calculus [9]. Whenever the agent moves, it updates the White Pages artifact. When an agent `a` calls the `.send` action in Jason, the action accesses the White Pages to check which other agents are in the same region as `a`.

In this section, we present the instantiation of KQML-S inside the VEsNA framework. The code is available to the community from the KQML-S repository [17].

VEsNA [20] extends Jason by enabling peer-to-peer connections between an agent and its body and uses some features of JaCaMo [4] when artifacts are needed. Jason, implemented in Java, provides the foundation for this extension. In Jason a class `Agent` manages the agent's lifecycle and reasoning process, interpreting the AgentSpeak(L) source, and agents have a predefined set of internal actions implemented through the `DefaultInternalAction` class. VEsNA extends the default Jason implementation in three ways:

1. the `Agent` class is extended into `VesnaAgent`, which creates a WebSocket client for bidirectional communication with the body;

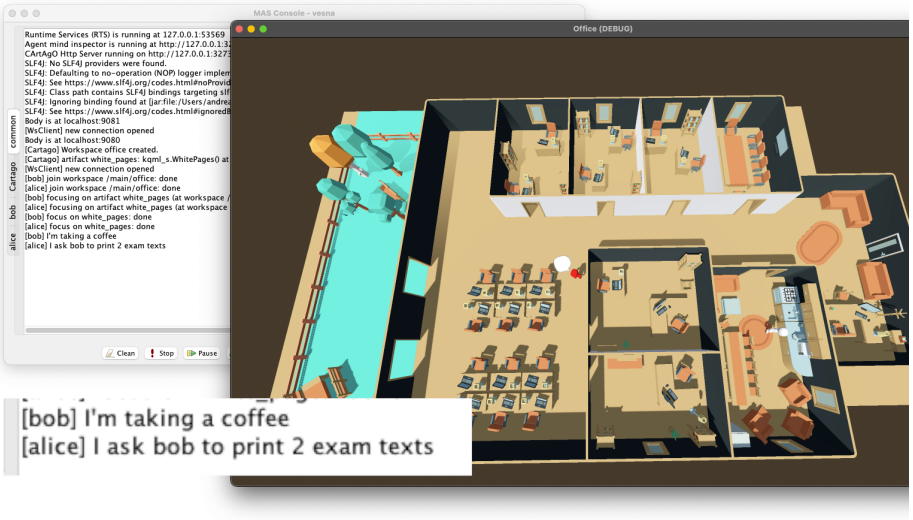

Fig. 3. Scenario 1. Alice (red agent) asks Bob (white agent) to print two copies of the exam. The request is heard by nobody.

2. a new `DefaultInternalAction walk` is introduced, enabling agents to perform walking actions;
3. a set of high-level plans is provided to facilitate agent implementation.

VEsNA agents reason logically on space using the Region Connection Calculus (RCC) [16]. In RCC, the relation `ntpp(X, Y)` describes that *X is strictly contained in Y*. This property is crucial for the implementation, as it naturally expresses an agent `a`'s situatedness within a region `r` as `ntpp(a, r)`. In the current KQML-S in VEsNA implementation, we only consider situatedness in regions: logical rooms like `WhatsApp` are not yet supported. Figure 2 shows the architecture we implemented.

To implement the KQML-S extension a new `DefaultInternalAction .send` has been developed starting from the `.send` standard one: the KQML-S `.send` action provides this API

```
.send( Mode, Receiver, Performative, Cnt, Answer, Timeout )
```

where `Mode` can be set to either `private` or `public` to change the privacy level of the message and the other parameters are the same as the default one. The message is sent only if the sender and `Receiver` are in the same region. Since in this current implementation we only support region-based situatedness, the `Mode` actual parameter, that in our formalization is a couple `(Room, How)`, only needs to state the `How` (`private` or `public`), as the `Room` is computed automatically as "the RCC region in the Virtual Environment where the sender is" by taking advantage of the *White Pages* artifact. If `Mode` is `public` all the agents in the region that are neither `Receiver` nor in `Receiver` list will find the

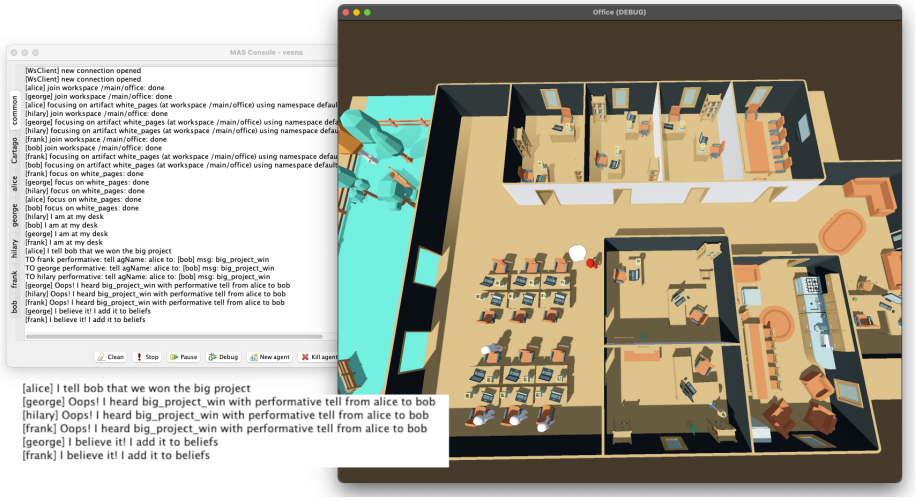

Fig. 4. Scenario 2. Alice (red agent) tells Bob (one of the white agents) they won a big project. The message is heard also by Frank, George and Hilary who are in the room. As shown in the MAS Console, Frank and George are credulous and they add the content to their belief base while Hilary does nothing.

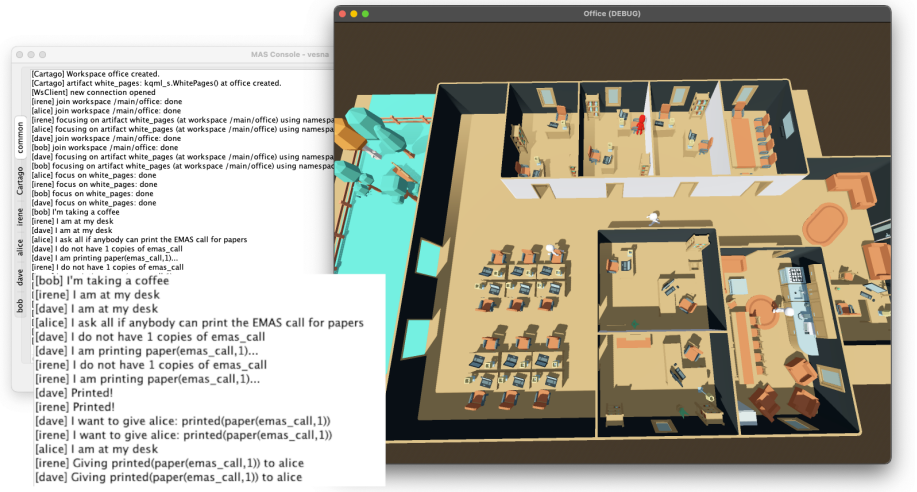

Fig. 5. Scenario 5. Alice (red agent) requests `all` agents inside the office room to print EMAS call for papers. The message is received from Bob and Irene, and both print it.

`notify( Performative, Receiver, Cnt )[source( Sender )]` belief among their mental notes. In addition, an agent can perform a `.send` with `all` as `Receiver`. In this case, the message is sent to all the agents present in the region.

Every time one agent moves from one region to another, it calls the *White Pages* artifact to update its position. The `.send` action retrieves all agents in the same room as the sender and delivers the message to them only. By leveraging this environmental artifact, locality information is correctly managed within the environment, preserving individual agent perspectives: one agent may believe that another agent is in a room based on prior information, or remain unaware of other agents' presence unless explicitly informed.

The implementation has been tested in the five scenarios described in Section 2, apart from Scenario 3 that considers logical rooms (`WhatsApp`), not supported yet. Scenario 4 has been implemented but is not discussed for space constraints. The agents are embodied in a virtual office environment implemented using Godot. The screenshots below capture a single moment of execution, while the MAS Console window provides a full log of previous events along with detailed real-time information on agent interactions.

Scenario 1. Alice moves to the open area of the office, where Bob is usually present, to request that he prints two copies of the exam. Alice performs `.send(public, bob, achieve, print(paper(exam, 2)));` and Bob has indeed a plan to print but, as shown in Figure 3, he is enjoying a coffee in the common room.

Scenario 2. Alice returns to the open office where Bob is and announces that they won a big project, speaking loudly without realizing that Frank, George, and Hilary are also present, as shown in Figure 4.

Scenario 5. Alice enters the office and asks if anyone can print a copy of the EMAS 2025 call for papers. In the office there are Bob and Irene, that she does not know. Alice sends the request using `send` with `all` as the receiver, ensuring that everyone in the room directly receives the `achieve` message. Since, in this simplified setting, no coordination mechanism is in place, both Bob and Irene print the document and provide a copy to Alice, as illustrated in Figure 5.

6 Related and Future Work

This paper introduced KQML-S, a KQML extension that incorporates locality and distinguishes between private and public communication, enabling agents to adjust interactions based on shared logical spaces. Inspired by realistic interaction dynamics such as overhearing and eavesdropping, KQML-S enriches coordination and awareness in MAS.

Situated communication in BDI-based frameworks has received limited attention, particularly in recent years. Our interest in this direction emerged from work on VEsNA, where agents act in virtual environments where locality shapes communication. As distributed systems increasingly rely on spatial or logical co-location, we expect locality-aware communication to grow in importance.

Previous approaches have touched on related aspects. CG-KQML+ [7] extends KQML with conceptual graphs and enhanced performatives, but lacks the focus on spatial context. Coo-BDI [1] enables plan sharing among trusted peers, emphasizing structural cooperation rather than dynamic, context-driven communication. Moise+ [24] models role-based social behavior, while KQML-S focuses on how locality and presence influence message propagation. Liu et al. [27] offer a domain-specific KQML adaptation for control systems, whereas KQML-S remains general-purpose. Dell'Anna et al. [10] address norm revision, which aligns with our interest in context-sensitive message interpretation.

Unlike these works, KQML-S introduces a communication layer grounded in environment-aware semantics and agent attitudes (e.g., altruistic, skeptical), supporting nuanced responses based on message content and context.

Looking forward, several directions can expand KQML-S. Extending the set of KQML performatives to include `Ask` or `TellHow` would support more expressive agent dialogues, enabling agents to handle a broader range of communication behaviors. Moreover, we aim to explore the impact of *nested* or *connected logical rooms* on message propagation, beyond discrete, isolated spaces. Additionally, integrating *social structures* such as *norms*, *roles*, and *hierarchies* would enable agents to adjust their communication strategies based on authority, trust, or group membership. Studying the connections between situated communication and *Information Protocols* like BSPL [38] would also expand the application field of the proposed approach.

Another important future direction is the integration of *coordination* and *conflict resolution protocols* to enhance the robustness of KQML-S. This would be especially useful in scenarios where multiple agents respond to a single overheard message, potentially leading to conflicts (e.g., multiple agents pursuing the same goal). Large-scale experiments will also be essential to assess how KQML-S scales in more complex MAS environments, testing its performance and scalability as the number of agents and logical spaces increases.

Looking beyond virtual environments, extending KQML-S to interact with physical environments and IoT devices could open new applications in domains like smart homes, autonomous vehicles, or robotic swarms. This would allow agents to interpret and react to both digital and physical cues in a more integrated, real-world context.

Finally, integrating KQML-S with privacy-preserving communication mechanisms, such as encryption or access control, could enhance its security. This would allow agents to reason not only about who might overhear a message but also about whether certain communications should be hidden, thereby bridging the gap between situated communication and security-aware MAS design.

These directions aim to evolve KQML-S into a comprehensive framework for situated communication in dynamic, socially rich MAS environments.

Acknowledgements. This work was partially supported by *ENGINES – EN-Gineering INtElligent Systems around intelligent agent technologies*, funded by the Italian MUR program PRIN 2022 under grant number 20229ZXBZM.

References

1. Ancona, D., Mascardi, V.: Coo-BDI: Extending the BDI model with cooperativity. In: Leite, J.A., Omicini, A., Sterling, L., Torroni, P. (eds.) Declarative Agent Languages and Technologies, First International Workshop, DALT 2003, Melbourne, Australia, July 15, 2003, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 2990, pp. 109–134. Springer (2003). https://doi.org/10.1007/978-3-540-25932-9_7, https://doi.org/10.1007/978-3-540-25932-9_7
2. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. John Wiley & Sons (2007)
3. Bergenti, F., Caire, G., Monica, S., Poggi, A.: The first twenty years of agent-based software development with JADE. Auton. Agents Multi Agent Syst. **34**(2), 36 (2020)
4. Boissier, O., Bordini, R., Hubner, J., Ricci, A.: Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo. Intelligent Robotics and Autonomous Agents series, MIT Press, United States (2020), https://books.google.com.br/books?id=GM_tDwAAQBAJ
5. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence - From Natural to Artificial Systems. Santa Fe Institute Studies in the Sciences of Complexity, Oxford University Press (1999)
6. Bordini, R., Hübner, J., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason, vol. 8. John Wiley & Sons, Ltd, United Kingdom (10 2007). https://doi.org/10.1002/9780470061848
7. Bouzouba, K., Moulin, B., Kabbaj, A.: CG-KQML+: an agent communication language and its use in a multi-agent system. In: Proc of the 9th Int. Conf. on Conceptual Structures. pp. 1–14 (2001)
8. Cohen, P.R., Perrault, C.R.: Elements of a plan-based theory of speech acts. Cognitive Science **3**(3), 177–212 (1979). https://doi.org/https://doi.org/10.1016/S0364-0213(79)80006-3, https://www.sciencedirect.com/science/article/pii/S0364021379800063
9. Cohn, A.G., Bennett, B., Gooday, J., Gotts, N.M.: Qualitative spatial representation and reasoning with the region connection calculus. GeoInformatica **1**(3), 275–316 (1997). https://doi.org/10.1023/A:1009712514511, https://doi.org/10.1023/A:1009712514511
10. Dell'Anna, D., Dastani, M., Dalpiaz, F.: Runtime revision of sanctions in normative multi-agent systems. Auton. Agents Multi Agent Syst. **34**(2), 43 (2020)
11. Ferrando, A., Winikoff, M., Cranefield, S., Dignum, F., Mascardi, V.: On enactability of agent interaction protocols: Towards a unified approach. In: EMAS@AAMAS. Lecture Notes in Computer Science, vol. 12058, pp. 43–64. Springer (2019)
12. Ferrando, A., Winikoff, M., Cranefield, S., Dignum, F., Mascardi, V.: On enactability of agent interaction protocols: Towards a unified approach. In: AAMAS. pp. 1955–1957. International Foundation for Autonomous Agents and Multiagent Systems (2019)
13. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: Proceedings of the Third International Conference on Information and Knowledge Management. p. 456–463. CIKM '94, Association for Computing Machinery, New York, NY, USA (1994). https://doi.org/10.1145/191246.191322, https://doi.org/10.1145/191246.191322

14. Finin, T., McKay, D.P., Fritzson, R., McEntire, R.: KQML - A Language and Protocol for Knowledge and Information Exchange. IOS Press (September 1994), revised paper from the 13th Int. Distributed Artificial Intelligence Workshop, July 28-30, 1994
15. Foundation for Intelligent Physical Agents: FIPA ACL Message Structure Specification (2002), http://www.fipa.org/specs/fipa00061/SC00061G.html, accessed on April 22, 2025
16. Gatti, A.: Reason logically, move continuously. In: Ferrando, A., Cardoso, R.C. (eds.) Agents and Robots for reliable Engineered Autonomy. pp. 115–127. Springer Nature Switzerland, Cham (2025)
17. Gatti, A., Ferrando, A., Mascardi, V.: KQML-S web site (2025), https://github.com/VEsNA-ToolKit/KQML-S, accessed on April 22, 2025
18. Gatti, A., Ferrando, A., Mascardi, V.: VEsNA-Toolkit web site (2025), https://github.com/VEsNA-ToolKit, accessed on April 22, 2025
19. Gatti, A., Mascardi, V.: Towards VEsNA, a framework for managing virtual environments via natural language agents. In: AREA@IJCAI-ECAI. EPTCS, vol. 362, pp. 65–80 (2022)
20. Gatti, A., Mascardi, V.: VEsNA, a framework for virtual environments via natural language agents and its application to factory automation. Robotics **12**(2), 46 (2023)
21. Genesereth, M.R., Ketchpel, S.P.: Software agents. Commun. ACM **37**(7), 48–ff. (Jul 1994). https://doi.org/10.1145/176789.176794, https://doi.org/10.1145/176789.176794
22. Godot foundation: Godot web site (2025), https://godotengine.org/, accessed on April 22, 2025
23. Hadeli, Valckenaers, P., Kollingbaum, M., Van Brussel, H.: Multi-agent coordination and control using stigmergy. Comput. Ind. **53**(1), 75–96 (Jan 2004). https://doi.org/10.1016/S0166-3615(03)00123-4, https://doi.org/10.1016/S0166-3615(03)00123-4
24. Hübner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the MOISE$^+$ model: programming issues at the system and agent levels. Int. J. Agent Oriented Softw. Eng. **1**(3/4), 370–395 (2007). https://doi.org/10.1504/IJAOSE.2007.016266, https://doi.org/10.1504/IJAOSE.2007.016266
25. Labrou, Y., Finin, T.: A semantics approach for KQML – a general purpose communication language for software agents. In: Proceedings of the Third International Conference on Information and Knowledge Management. p. 447–455. CIKM '94, Association for Computing Machinery, New York, NY, USA (1994). https://doi.org/10.1145/191246.191320, https://doi.org/10.1145/191246.191320
26. Labrou, Y., Finin, T.: A proposal for a new KQML specification. Tech. rep., TR CS-97-03 from UMBC (1997)
27. Liu, Y., Zhang, X., Wu, Q.: Optimizing KQML for usage in wood drying multi-agent coordination system. In: 2011 Chinese Control and Decision Conference (CCDC). pp. 3725–3730 (2011). https://doi.org/10.1109/CCDC.2011.5968872
28. Luke, S., Balan, G.C., Sullivan, K., Panait, L.: Mason (2003), https://cs.gmu.edu/~eclab/projects/mason/, george Mason University
29. North, M.J., Collier, N.T., Vos, J.R.: Repast (2006), https://repast.github.io/, argonne National Laboratory
30. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Auton. Agents Multi Agent Syst. **17**(3), 432–456 (2008)
31. Omicini, A., Zambonelli, F.: Coordination for internet application development. Auton. Agents Multi Agent Syst. **2**(3), 251–269 (1999)

32. Pokahr, A., Braubach, L.: From a research to an industry-strength agent platform: JADEX V2. In: Wirtschaftsinformatik (1). books@ocg.at, vol. 246, pp. 769–780. Österreichische Computer Gesellschaft (2009)
33. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996. Lecture Notes in Computer Science, vol. 1038, pp. 42–55. Springer (1996). https://doi.org/10.1007/BFb0031845, https://doi.org/10.1007/BFb0031845
34. Ricci, A., Omicini, A., Denti, E.: The TuCSoN coordination infrastructure for virtual enterprises. In: Proceedings Tenth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. WET ICE 2001. pp. 348–353 (2001). https://doi.org/10.1109/ENABL.2001.953442
35. Ricci, A., Piunti, M., Viroli, M., Omicini, A.: Environment programming in CArtAgO. In: Multi-Agent Programming, Languages, Tools and Applications, pp. 259–288. Springer (2009)
36. Ricci, A., Viroli, M., Omicini, A.: Give agents their artifacts: the A&A approach for engineering working environments in MAS. In: AAMAS. p. 150. IFAAMAS (2007)
37. Singh, M.P.: Agent communication languages: Rethinking the principles. Computer **31**(12), 40–47 (1998). https://doi.org/10.1109/2.735849, https://doi.org/10.1109/2.735849
38. Singh, M.P.: Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 491–498 (2011)
39. Theraulaz, G., Bonbeau, E.: A brief history of stigmergy. Artif. Life **5**(2), 97–116 (Apr 1999). https://doi.org/10.1162/106454699568700, https://doi.org/10.1162/106454699568700
40. Vieira, R., Moreira, Á.F., Wooldridge, M.J., Bordini, R.H.: On the formal semantics of speech-act based communication in an agent-oriented programming language. J. Artif. Intell. Res. **29**, 221–267 (2007). https://doi.org/10.1613/jair.2221, https://doi.org/10.1613/jair.2221
41. Wilensky, U.: Netlogo (1999), ttp://ccl.northwestern.edu/netlogo/, center for Connected Learning and Computer-Based Modeling, Northwestern University