Ontological Overhearing[⋆]

Marco Aiello[†], Paolo Busetta[*], Antonia Donà[*], and Luciano Serafini[*]

[†] ILLC and ISIS, University of Amsterdam
Plantage Muidergracht 24, 1018 TV Amsterdam, the Netherlands
[†] DISA - University of Trento, Via Inama 5, 38100 Trento, Italy
[*] ITC-IRST, Via Sommarive 18, 38050 Povo, Trento, Italy

Abstract. The collaboration between two intelligent agents can be greatly enhanced if a third agent, who has some understanding of the communication between the first two, intervenes giving appropriate information or acting helpfully without having been explicitly involved. The behavior of this third agent, quite common in human interaction, is called *overhearing*. We present an agent architecture modeling this behavior. In particular, we focus on overhearing based on ontological reasoning; that is, the overhearer semantically selects pieces of communication according to his own knowledge (ontologically organized) and goals. In our architecture, overhearing is performed by a team of agents playing two different roles: the first role (overhearer) classifies the overheard communication according to a formal ontology; the second role (suggester) makes appropriate suggestions at the appropriate time point. We present a formal language for the interaction between agents in the overhearing team. A prototype of the architecture, implemented using JACK Intelligent Agents, is briefly described and preliminary experimental results are discussed.

1 Introduction

Humans work well in teams. In a collaborative environment, whenever people are faced with tasks that they cannot manage, or know that can be managed better by others, they seek assistance. This observation is not new, and has inspired much research in cooperative agents, for example [13, 18, 17].

We choose to analyze teamwork through a slight shift in perspective. While association between agents can readily be achieved by requesting help when needed, equal or even improved results can be achieved when associates observe the need for help, and initiate actions or offer suggestions with the aim of improving the plight of their colleague. In fact, these associates may communicate not only when a colleague needs assistance, but also when they feel that they can help improve the productivity of their team, or when they believe to achieve a personal gain from the suggestion.

As part of our work on frameworks for collaboration, we have introducted an abstract architecture based on a principle called *overhearing* [10]. The intuition behind overhearing comes from the modeling of such human interaction as aforementioned,

[⋆] We thank Floriano Zini for his participation in the definition of the architecture, and the discussions on $\mathcal{PT}_{\mathcal{ALC}}$. Special thanks are due to ITC-IRST, to the University of Trento and to Prof. Fausto Giunchiglia for supporting the extended visit by Marco Aiello that lead to this research.
Contact Author: Luciano Serafini, e-mail `serafini@itc.it`, Tel +39 0461 314319, ITC-IRST Via Sommarive 18, 38050 Povo, Trento, Italy

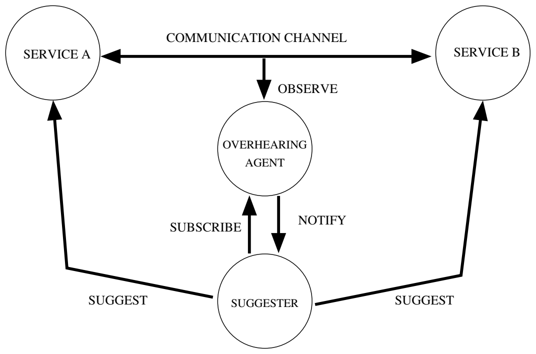

Fig. 1. Overhearing architecture.

in a collaborative observable environment. The overhearing architecture describes how non-planned collaboration in a community of artificial agents can be achieved by means of *unobtrusive observations* and *unsolicited suggestions*.

This paper focuses on a single aspect of the architecture: the observation of the conversations between two or more agents. Our goal is to provide a framework for querying a communication channel on the development of a conversation; queries are typically submitted by agents willing to provide unsolicited assistance. To this end, we have defined a formal language, designed the software components required for its interpretation, and performed some experiments on a real-life, multi-agent, observable channel (a web-based newsgroup). Our formal language includes both temporal and ontological components, and allows the formulation of complex queries on contents, performatives, and order of the messages being exchanged.

The paper is organized as follows. Next section gives an overview of the overhearing architecture. In Section 3 we provide the definition of the language for querying the communication channel, and show how formulas of the language are interpreted. In Section 4, we present the use of the formal language within the overhearing architecture. An actual implementation and some experimental results, showing the effectiveness of the proposed framework, are presented in Section 5. Section 6 discusses some related work. We conclude identifying potential future research (Section 7).

2 The Overhearing Architecture: an Overview

The overhearing abstract architecture is summarized in Figure 1. Service Agent A and Service Agent B are communicating over a channel and observed by the Overhearing Agent (*overhearer* from here on). A Suggester Agent (*suggester*) subscribes with the overhearer to be notified if a certain type of event has occurred on the channel. Finally, the Suggester is able to issue *suggestions* to any of the service agents; a suggestion is a special message carrying information or commands. In general, a running system contains many couples of agents covering the role of services, more than one agent acting as suggesters, and at least one overhearer; an agent may cover more than one role simultaneously (such as service and suggester).

Overhearing differs from blackboard-based architectures (see, for instance, the Open Agent Architecture [11]) because it is not concerned with connecting services, nor controlling the flow of messages. Our aim is not providing yet another communication facility, rather supporting a flexible development methodology for complex, adaptive systems. In the initial phases of development, only those agents (services in the terminology defined above) required to achieve the basic functionality should be built. The behavior of these services, however, should be modifiable by external observers via suggestions. While functionality of the basic services required by an application are assumed to be immutable, suggesters may be added and removed dynamically without hampering the ability of the system to reach its main objectives.

This approach has various advantages. Firstly, it is possible to enhance functionality of a running system. As an example, state-of-the-art machine-learning or tunable components can be plugged into the system, as and when they become available, without the need to bring down, rebuild, and then restart the system. Secondly, the output of a system can be enhanced either by suggesting additional related information, or requesting the deletion of outdated or unrelated results. In [10] we present, as a case study, an agent-based Web server where suggesters can send additional data to assistant agents building dynamic HTML pages from databases of museum collections and other cultural information. Schema and contents of these databases are stable over time. Temporary facts (e.g., exhibitions) and data that cannot easily be fit into an relational database (e.g., unstructured text, targeted advertising based on dynamically built user profiles) are collected by suggesters, and sent to the assistants whenever appropriate.

The flexibility offered by the overhearing architecture comes at a cost. Indeed, a full implementation of the overhearing architecture requires the suggester to be able to perform agent state recognition, based on a model of the service agents and the messages they exchange. Communication – or at least some selected conversations – needs to be observable, e.g., by using a broadcast service, and this in turn may introduce issues with performance, timing and security. Last but not least, services need to be engineered to handle suggestions and to change their behavior accordingly.

In what follows, we concentrate on the interaction between suggester and overhearer. As mentioned above, its objective is for suggester to be notified of all and only those messages that are relevant to him.

3 Overhearer–Suggester Interaction Language

In the architecture presented above, an overhearer has two main goals: monitoring a communication channel, and responding to the queries of suggester agents about the conversations taking place.

Depending on the application, messages exchanged by services may vary from fully structured (e.g., most client/server interactions, auctions, etc.), to semi structured (e.g., XQL queries and XML pages), to unstructured (e.g., the body of email or newsgroup messages). Often, content is in natural language; sometimes it may even be of a multimedia nature such as images and sound tracks. In many situations, it is not feasible for an overhearer to keep track of the entire content of all messages. In case of intensive communication, indeed, this simply requires too much memory space. An overhearer keeps track of the conversations by logging a suitable amount of data for each message traveling on the channel, that is a “summary” representing an *interpretation* of the message with respect to a certain formal domain ontology.

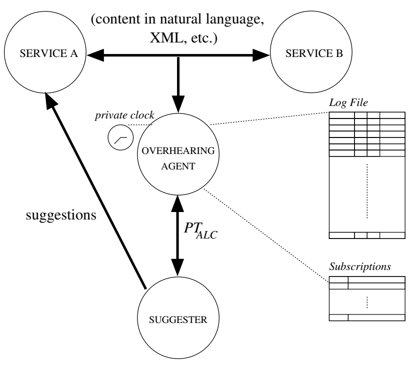

Fig. 2. Overhearing architecture: the interaction language and facilities of the overhearer.

Suggesters that want to be notified of the messages regarding specific topics can subscribe to the overhearer. In its subscription, a suggester specifies a matching criteria (a pattern), which the overhearer has to apply to select the messages to be forwarded to the suggester.

To provide a reliable and well founded tool for expressing such a selection criteria, we developed a multi-modal language inspired by description logics and modal temporal languages. We called the language $\mathcal{PT}_{\mathcal{ALC}}$ since it is a modal $\mathcal{T}$emporal logic over a simple description language of the $\mathcal{ALC}$ family enriched with $\mathcal{P}$erformatives. Figure 2 highlights the use of $\mathcal{PT}_{\mathcal{ALC}}$ for subscriptions, and the main facilities used by the overhearer: a list of subscriptions, an internal clock to time the channel, and a log file for the messages. As discussed later, the latter contains message interpretations in $\mathcal{P}_{\mathcal{ALC}}$, which is a subset of $\mathcal{PT}_{\mathcal{ALC}}$ without temporal operators.

3.1 A temporal ontological language

Suppose a suggester's goal is to bring a certain castle, the Buonconsiglio Castle in Trento, to the attention of potential tourists of the Trentino region. This can be achieved in various ways. For instance, the suggester could ask to be informed of all the messages passing on a public channel between an agent browsing the web and an agent serving web pages; or, he could ask to be informed of all the messages containing a specific set of words (e.g., {Castle, Trento, Tourism}). Alternatively, he could ask to be informed whenever a message containing concepts related to castles and tourism has been uttered. The latter appears to be the most appropriate approach, therefore we require the overhearer-suggester interaction language to be able to express structured concepts. In addition, a temporal dimension is necessary to express properties of the temporal order of messages.

The suggester can ask if an agent has uttered a request regarding castles located in Trentino, or regarding a particular castle located in Trentino, in the following way:

$$\Diamond_p(\textsc{Ask}(*, *, castle \sqcap \forall is\_located.Trentino))$$

The suggester is expressing information of a very different nature with the above formula. Let us analyze it bottom-up:

- **Conceptual:** $\mathcal{ALC}$**.** $castle \sqcap \forall is\_located.Trentino$ The suggester is expressing the concept of things that are both castles and that for all roles 'is_ located' have a filler of type Trentino. We denote this formula by φ.
- **Performative:** $\mathcal{P}_{\mathcal{ALC}}$**.** $\textsc{Ask}(*,*,\varphi)$ The suggester is interested in performance of the type ASK from any agent to any other agent. The $*$ stands for a wild-card, in alternative he could have explicitly referred to the name of agents involved in the channel. We denote this formula by ψ.
- **Temporal:** $\mathcal{PT}_{\mathcal{ALC}}$**.** $\diamond_p \psi$ The suggester is interested of performances ψ occurred at some point in the past, expressed by the temporal operator $\diamond_p$.

In the next three subsections, we give the precise syntax and semantics of all the pieces of the language $\mathcal{PT}_{\mathcal{ALC}}$.

The conceptual language Description logics are a family of formalisms spun off from research in semantic networks. One of the main advantages of description logic formalisms over previous approaches is the availability of a precisely defined semantics which ensures correctness of reasoning tasks such as subsumption checking. In what follows, we use notation and semantics borrowed from the description logics community (see for instance [20, 5]).

The alphabet is composed of three main sorts: one for atomic concept names, one for role names and one for object names. For instance, $castle$ is an atomic concept name: that of all entities being castles. $is\_located$ is a role name: that connecting concepts to other concepts in which they are located. $Buonconsiglio$ is the name of an object: in this case, a physical castle located in Trento.

From the alphabet symbols it is possible to define the set of *concepts* as follows. Notationally, A represents an atomic concept name, C and D denotes any concept, and R denotes a role name.

$$C, D ::= A \mid C \sqcup D \mid C \sqcap D \mid \forall R.C \mid \exists R.C$$

Relations among concepts, and relations among objects and concepts can be expressed by means of two types of *formulas*:

- $o : C$ is an instance selection,
- $C \sqsubseteq D$ is a subsumption statement.

Formulas of $\mathcal{ALC}$, or equivalently $\mathcal{ALC}$-formulas are denoted by small Greek letters, ϕ, ψ, ... Using the syntax above, we can specify that $Buonconsiglio$ is an object (also referred to as *instance*) of the concept of things which are $castle$s and are located somewhere in the province of $Trentino$:

$$Buonconsiglio : castle \sqcap \forall is\_located.Trentino$$

We can also state that the concept of castle is less general than the concept of large building, by the formula

$$castle \sqsubseteq building \sqcap large$$

The semantics is given in terms of interpretations I. An interpretation is a pair composed of a *domain* Δ^I, and a *interpretation function* $\cdot^I$ that assigns a subset of Δ^I to each concept name, a subset of $\Delta^I \times \Delta^I$ to every role name, and an element of Δ^I to each object name. The interpretation I for all the concepts is defined as follows:

$$\begin{aligned}
(C \sqcup D)^I &= C^I \cup D^I \\
(C \sqcap D)^I &= C^I \cap D^I \\
(\forall R.C)^I &= \{p \in \Delta^I \mid \text{for all } q \in \Delta^I, \langle p,q\rangle \in R^I \text{ implies } q \in C^I\} \\
(\exists R.C)^I &= \{p \in \Delta^I \mid \text{there is a } q \in \Delta^I, \text{ such that } \langle p,q\rangle \in R^I \text{ and } q \in C^I\}
\end{aligned}$$

Formulas can be verified or falsified by an interpretation I according to the following rules:

$$I \models o : C \text{ if and only if } o^I \in C^I$$
$$I \models C \sqsubseteq D \text{ if and only if } C^I \subseteq D^I$$

A *Terminological box*, (*TBox*) is a set of subsumption statements; an *assertional box* (*ABox*) is a set of instance selections. A *TBox* together with an *ABox* form a *knowledge base*. We use Σ to denote a knowledge base. An interpretation is a *model* of a knowledge base Σ if every sentence of Σ is satisfied in the interpretation.

The performative language The performative level of the language allows to express intentions in connection with concepts. In general, such intentions can be of very different sorts: asking a question, replying to a comment, requesting an action, etc. In general, we identify a set of performatives Per, which represents all performances that can be recognized by the overhearer from the messages passing on the channel. For the sake of simplicity, we concentrate on the two performatives ASK and TELL, representing the performance of a query and the performance of an assertion, respectively. The set of *message patterns* of $\mathcal{P}_{\mathcal{ALC}}$ is any object of the form:

$$\textsc{Per}(i, j, \phi) \tag{1}$$

where PER is either ASK or TELL, i, j are either names of agents or the wild-card $*$, and ϕ is either an object name or a concept, or an instance selection, or a subsumption statement. Message patterns are denoted by small Greek letters μ, ν. We also denote the set of agent names with Ag. Message patterns that does not contains wild-cards are called *messages*.

Intuitively, the message pattern $\textsc{Ask}(i, *, C)$ represents the set of messages which are queries posted by agent i to any other agent, concerning the concept C. The formula $\textsc{Ask}(i, *, o : C)$ represent the set of messages which are queries posted by agent i to any other agent, concerning the object o as an instance of a concept C. To formalize this intuition we need to extend the semantics of $\mathcal{ALC}$. Formally: an interpretation I of $\mathcal{ALC}$ can be extended to an interpretation for $\mathcal{P}_{\mathcal{ALC}}$ in the following way:

$$\begin{aligned}
i^I &= \{i\} \text{ if } i \text{ is not a wild-card; } Ag \text{ otherwise} && (2)\\
(\textsc{Per}(i,j,o))^I &= \{\langle \textsc{Per}, x, y, o^I\rangle \mid x \in i^I \text{ and } y \in j^I\} && (3)\\
(\textsc{Per}(i,j,C))^I &= \{\langle \textsc{Per}, x, y, D^I\rangle \mid x \in i^I \text{ and } y \in j^I \text{ and } I \models D \sqsubseteq C\} && (4)\\
(\textsc{Per}(i,j,o:C))^I &= \{\langle \textsc{Per}, x, y, o^I, D^I\rangle \mid x \in i^I \text{ and } y \in j^I \text{ and } I \models D \sqsubseteq C\} && (5)\\
(\textsc{Per}(i,j,C \sqsubseteq D))^I &= \{\langle \textsc{Per}, x, y, C^I, E^I\rangle \mid x \in i^I \text{ and } y \in j^I, I \models E \sqsubseteq D\} && (6)
\end{aligned}$$

Subsumption between message patterns is defined in terms of containment of their interpretation. Formally:

$$I \models \mu \sqsubseteq \nu \text{ if and only if } \mu^I \subseteq \nu^I \tag{7}$$

Intuitively $I \models \mu \sqsubseteq \nu$ means that, according to the interpretation I, any message that matches patterns μ match also pattern ν; put differently, the pattern μ is more specific (less general) than the pattern ν.

The full temporal language $\mathcal{PT}_{\mathcal{ALC}}$ The last channel phenomenon we want to model is time. We chose the same formalism of Since and Until logics (see for instance [21]), which gives us the ability to refer to the past, to the future, to the next temporal interval (the next utterance) but also to place conditions on future or past events.

The syntax of the temporal language $\mathcal{PT}_{\mathcal{ALC}}$ is as for the previous languages, with the addition of the following operators over formulas of $\mathcal{P}_{\mathcal{ALC}}$: $\neg$, X_f, $\diamond_f$, $\Box_f$, X_p, $\diamond_p$, $\Box_p$ as temporal monadic operators, $\mathcal{S}, \mathcal{U}$ (since and until) as temporal dyadic operators and the classical connectives. $\wedge$, $\vee$, and $\neg$. Operators labelled with a p $[f]$ refers to the past [future]. Rather than over propositional letters, all these operators work over message patterns. Therefore, the atomic formulas of $\mathcal{PT}_{\mathcal{ALC}}$ are message patterns. Intuitively the atomic formula composed of the message pattern μ means that a message that matches μ is now passing on the channel. The formula $\mu \wedge \nu$ means that a message that matches both μ and ν is now passing on the channel. The formula $X_p\mu$ means that a message that matches the pattern μ has passed on the channel at the previous time stamp. Formulas in $\mathcal{PT}_{\mathcal{ALC}}$ are denoted by Greek letters α, β, etc. Notice that any message pattern μ is a formula (actually an atomic formula) of $\mathcal{P}_{\mathcal{ALC}}$.

Since the objects of our temporal logic are performatives over message patterns, rather than propositional letters that can be true or false at a given time point as in usual logics, we need to specify the semantics for $\mathcal{PT}_{\mathcal{ALC}}$ in terms of Kripke structures on interpretations of message patterns. This semantics is an extension of the Kripke-like semantics for since and until logics in the case of linear discrete time, bounded in the past.

A model M or $\mathcal{PT}_{\mathcal{ALC}}$ on the knowledge base Σ is an infinite sequence

$$M = M(1), M(2), \ldots$$

indexed by natural numbers, where for each natural number s, $M(s)$ is a pair $\langle \mu(s), I(s) \rangle$, composed of a message $\mu(s)$ and an interpretation $I(s)$ of $\mathcal{ALC}$ that is a model of Σ. Satisfiability for $\mathcal{PT}_{\mathcal{ALC}}$ is defined as follows:

$M, s \models \nu$	iff $I(s) \models \mu(s) \sqsubseteq \nu$	$M, s \models \neg\alpha$	iff $M, s \not\models \alpha$
$M, s \models \alpha \vee \beta$	iff $M, s \models \alpha$ or $M, s \models \beta$	$M, s \models \alpha \wedge \beta$	iff $M, s \models \alpha$ and $M, s \models \beta$
$M, s \models X_f\alpha$	iff $M, s+1 \models \alpha$	$M, s \models X_p\alpha$	iff $s > 0$ and, $M, s-1 \models \alpha$
$M, s \models \diamond_f\alpha$	iff for some $t > s$, $M, t \models \alpha$	$M, s \models \diamond_p\alpha$	iff for some $t < s$, $M, t \models \alpha$
$M, s \models \Box_f\alpha$	iff for all $t > s$, $M, t \models \alpha$	$M, s \models \Box_p\alpha$	iff for all $t < s$, $M, t \models \alpha$

$M, s \models \alpha\mathcal{S}\beta$ iff for some $t < s$, $M, t \models \beta$, and for all $t \leq w < s$, $M, w \models \alpha$
$M, s \models \alpha\mathcal{U}\beta$ iff for some $t > s$, $M, t \models \beta$, and for all $s \leq w < t$, $M, w \models \alpha$

Intuitively, the truth condition $M, s \models \nu$ represents the fact that the message passed at time s (i.e., $\mu(s)$) matches the message pattern ν, according to the current interpretation of conceptual language given by the overhearer (i.e., $I(s) \models \mu(s) \sqsubseteq \nu$). Connectives are treated classically; truth conditions for temporal operators are the usual conditions for linear past and future temporal logic bounded in the past.

4 Answering Queries from the Suggester

We now focus on the use of $\mathcal{PT}_{\mathcal{ALC}}$ by overhearer and suggester. When the suggester needs to be informed of something regarding the communication among service agents, he formulates the knowledge he wants to gain in terms of a $\mathcal{PT}_{\mathcal{ALC}}$ message pattern. Suppose that the suggester wants to be notified when an agent i has asked about castles in Trentino, and nobody has answered mentioning the existence of the Buonconsiglio Castle. The suggester would then *subscribe* to the overhearer with the following pattern:

$$\begin{array}{l}\Diamond_p(\textsc{Ask}(i, *, \mathit{castle} \sqcap \forall \mathit{is\_located}.\mathit{Trentino})) \\ \ \mathcal{U}\ (\textsc{Tell}(*, i, \mathit{Buonconsiglio}))\end{array} \tag{8}$$

The intended meaning of this subscription is to *notify the subscribing agent at every time instant for which the subscribed formula is true.* If, for instance, agent i at time t has performed $\textsc{Ask}(i, j, \mathit{castle})$ and at time $t + 1$ no agent has performed a $\textsc{Tell}$ to i with $\mathit{Buonconsiglio}$ as content, then at time $t + 1$ formula (8) becomes true and the overhearer will notify the suggester.[1] One may imagine that the suggester would then directly send a message to the agent i at time $t + 2$, informing of the existence of the Buonconsiglio Castle in Trentino. The informative action of the suggester, if done over the same communication channel observed by the overhearer, would also make formula (8) false at all times greater than $t + 2$, so he will not be notified again. The question is now, how does the overhearer know if and when a given $\mathcal{PT}_{\mathcal{ALC}}$ formula is true?

4.1 Checking the log

As shown in Figure 2, the overhearer updates two main data structures: a log file, storing information about the communication on the channel, and a file of the active subscriptions from the various agents.

In order to interpret the messages he logs, the overhearer needs to use an *ontology*. In the following, we assume that this ontology is a knowledge base Σ as described in Section 3.1.

The log file has a structure similar to the following:

Key	Time	Sender	Receiver	Performative	Content in φ
1	12:38:59 PM	i	j	$\textsc{Ask}$	castle
2	12:39:07 PM	j	i	$\textsc{Tell}$	$\mathit{Buonconsiglio} : \mathit{castle}$
3	12:39:08 PM	k	j	$\textsc{Ask}$	$\mathit{wheather} \sqcap \mathit{forecast}$
.	...	.	.	...	...

and it is updated every time that a performance has occurred on the channel. The active subscription file is simply a list of couples with a suggester's name and a $\mathcal{PT}_{\mathcal{ALC}}$ subscribed formula, updated every time a suggester subscribes or unsubscribes.

[1] Note the subsumption of the concept of the castles located in Trentino by the more general concept of a castle, i.e., $\Sigma \models \mathit{castle} \sqcap \forall \mathit{is\_located}.\mathit{Trentino} \sqsubseteq \mathit{castle}$.

The log file closely resemble a model for $\mathcal{PT}_{\mathcal{ALC}}$. Indeed, we have a family of ψ messages indexed by natural numbers bound both in the past and in the future; if we associate the ontology Σ with each formula ψ and think of an infinite future (just add $\top$ as formula to every future time step), we get a model for $\mathcal{PT}_{\mathcal{ALC}}$. The overhearer uses this model to check whether any subscribed formula is true at the current time point. The procedure to follow when a new message has been observed is straightforward:

1. interpret the message using Σ, and update the log file;
2. for all subscribed formulas:
   (a) check if it is true in the current time step
   (b) if it is, notify the corresponding suggester

Various optimizations to the algorithm are immediate. If a formula is never going to be true in the future (see the example at the beginning of this section), never check it again. If a formula refers to the past (e.g., $\Diamond_p$) keep a trace of its relevant truth values in the past time instants, instead of recalculating the truth at each time step. The overhearer could try to check the truth of a formula at each time step incrementally, just by checking if a new message has modified its truth value.

5 From Theory to Practice

We ran some experiments to verify that $\mathcal{PT}_{\mathcal{ALC}}$ is an adequate language of interaction between overhearer and suggester. More specifically, we wanted to see if the expressive power of $\mathcal{PT}_{\mathcal{ALC}}$ is sufficient and, at the same time, not too powerful, so to enable the suggester to 'hear' all the messages, and only those, in which he is interested.

We prototyped the overhearer–suggester interaction using JACK Intelligent Agents [2], a Java-based BDI platform. The overhearer was equipped with a parser, developed with JavaCC [3], for a subset of the performative language $\mathcal{P}_{\mathcal{ALC}}$, which was chosen as the core of the complete $\mathcal{PT}_{\mathcal{ALC}}$. The overhearer had an ontology, in the form of a set of beliefs, implementing a knowledge base as described in Section 3.1. Finally, the overhearer was provided with some weak natural language processing (NLP) capabilities (see for instance [7]); most notably, a list of stop-words for Italian.

In order to implement the matching conditions expressed in equations (2–6) we developed algorithms for instance checking, for checking the identity of instances, and for subsumption checking. The first two are rather trivial; the interesting portion is the subsumption checking. Traditionally, there are two ways to tackle this problem: in the syntactic approach, tableau methods are used (see for instance [16]); in the semantic approach, it is necessary to build a so called 'description graph' representing the model of the formula with respect to the knowledge base. We chose to follow the latter, extending the algorithm of Borgida and Patel-Schneider [9]. The extension was necessary because we allow concept disjunctions $\sqcup$. We embedded also the instances (not dealt with in [9]) in the graphs. Unlike the work in [9], we do not allow cyclic axiom definition. The modifications we made to the algorithm were possible since we deal with small ontologies and small formulas. In fact, the description graphs tied to $\mathcal{P}_{\mathcal{ALC}}$ grow exponentially in the size of the formula and also grow with the size of the ontology.

We tested the prototype on Italian newsgroups. A newsgroup can be seen as a communication channel between intelligent agents exchanging messages in natural language. We used the subjects of the messages as the content of the communications, interpreted by the overhearer according to its own ontology and transformed in formulas of $\mathcal{P}_{\mathcal{ALC}}$.

5.1 Experimental results

We launched the prototype on two newsgroups on an Italian web site, called Global News (`http://www.globalnews.it`). The topic of the first newsgroup is meteorology, while that of the second is traveling. We created two different ontologies: one about meteorological phenomena, consisting of 68 concepts (mainly divided in 4 categories: meteorological phenomena, time periods, political geography and orography) and 230 instances; a second one about traveling, made up of 190 concepts and 519 instances. We used a list of 544 Italian stop-words. The overhearer observed a total of 160 messages on the meteorological newsgroup and 754 messages on the traveling one.

The top table of Figure 3 contains ten subscription queries posed by suggesters to the overhearer. Questions Q1-Q4 were posed on the meteorological newsgroup using the meteorological ontology. For Q5, the same meteorological ontology was used, but on the traveling newsgroup. Finally, Q6-Q10 were posed on the traveling newsgroup with the traveling ontology. Even though the system allows for roles in the grammar, we did not used roles in the queries, because our ontologies were too simple.

The bottom tables of Figure 3 present the results of the queries under different assumptions. R is the number of messages that generated a notification event for the suggester (in parenthesis the number of wrong retrievals). T are the messages that were supposed to generate the notification (ground truth). Prc. and Rec. are the standard information retrieval measures [7] of precision and recall; specifically, Prc.=$\frac{|R \cap T|}{|R|}$ and Rec.=$\frac{|R \cap T|}{|T|}$. The last columns of the tables represent the two main causes of missing notification to suggesters: # ont. is the number of errors due to insufficient care in the design and implementation of the ontology, while # NLP is the number of errors due to misuse of NLP-information retrieval techniques.

The numbers in the right table were obtained by assuming that overhearer and the newsgroup members share the same ontology, i.e. same word, same meaning. For the left table, we assumed that this is not necessarily the case – "snow" can be either frozen water or a color (as in "snow white"). Since the messages contain no reference to an ontology and our NLP capabilities are very limited, we arbitrarily derived an instance concept by assuming that this was the closest word preceding it, following the Italian syntax – snow ("neve") in snow white ("bianco come la neve") would be an instance of white ("bianco").

The results shown in Figure 3 seem to demonstrate that our language has a discriminating power among the messages, and that the suggesters could express their wishes of notification without ambiguity. The only problematic case is the one of query Q5, where the structure of the meteorology ontology causes the very low value of precision. This is because some of the concepts related to geography and orography subsume part of the concepts on the traveling domain, so causing wrong notifications.

It should be noted that our aim was not that of building an information filtering system. If a robust system for interpreting natural language has to be developed, it would be necessary to employ better NLP tools. Presently only stop-words are removed; stemming techniques are also necessary, acronyms should be eliminated, given names should be handled, synonyms solved, and so on; some shallow parsing could also help. Also, the ontologies were rather small; for instance, the traveling ontology only contained geographical information of the main tourist resorts. To increase both precision and recall, richer ontologies should have been developed, but this was beyond the scope of the experimentation.

id.	Query	Translation
Q1	$\textsc{Ask}(*, *, \top)$	everything
Q2	$\textsc{Ask}(*, *, neve)$	snow
Q3	$\textsc{Ask}(*, *, vento \sqcup (mare \sqcup lago))$	wind or sea or lake
Q4	$\textsc{Ask}(*, *, precipitazioni \sqcap grafici)$	graphs of precipitation
Q5	$\textsc{Ask}(*, *, \top)$	everything
Q6	$\textsc{Ask}(*, *, \top)$	everything
Q7	$\textsc{Ask}(*, *, crociera \sqcup (Caraibi \sqcup Europa)$	cruising
Q8	$\textsc{Ask}(*, *, offerte)$	special offers
Q9	$\textsc{Ask}(*, *, alloggio)$	accommodation
Q10	$\textsc{Ask}(*, *, alloggio \sqcap (mare \sqcup Europa)$	accommodation in Europe or at sea

id.	R	T	Prc.	Rec.	# ont.	# NLP	R	T	Prc.	Rec.	# ont.	# NLP
Q1	66 (4)	82	$0.9\bar{3}$	0.756	6	14	83 (6)	82	0.928	0.939	5	0
Q2	14	16	1	0.875	1	1	14	16	1	0.875	1	1
Q3	11	12	1	0.917	0	0	12	12	1	1	0	0
Q4	0	2	UNDEF	0	0	0	2	2	1	1	0	0
Q5	23 (18)	5	0.217	1	0	0	153 (148)	5	$0.0\bar{3}$	1	0	0
Q6	306 (26)	358	0.915	0.782	48	32	357 (26)	358	0.927	0.925	12	15
Q7	32	129	1	0.248	0	0	103 (7)	129	0.932	0.744	29	4
Q8	7	7	1	1	0	0	7	7	1	1	0	0
Q9	26	38	1	0.684	0	0	27	38	1	0.711	5	6
Q10	10	30	1	$0.\bar{3}$	13	7	15	30	1	0.5	8	7

Fig. 3. Experimental results, left: different ontologies, right: shared ontology

In spite of the current limitations, $\mathcal{P}_{\mathcal{ALC}}$ is significantly more powerful than using simple pattern matching expressions for single words. For instance, the patterns corresponding to Q10 are generated by the cartesian product of the set of objects subsumed by ‘alloggio’ (accommodation, i.e. everything from hotels to camp sites) with the union of objects subsumed by ‘mare’ (sea) and ‘Europa’ (any European place defined in the overhearer’s ontology).

6 Some comparisons

It is worthwhile to highlight differences and relationships between $\mathcal{PT}_{\mathcal{ALC}}$ and agent communication languages (ACLs) such as FIPA-ACL [1] and KQML [14]. $\mathcal{PT}_{\mathcal{ALC}}$ is a logic language used by suggesters to query the log of messages kept by an overhearer in a way similar to, for instance, how database applications use SQL to interrogate a remote relational database. An ACL may be used to *transport* both the queries formulated in $\mathcal{PT}_{\mathcal{ALC}}$ and the messages exchanged by service agents. If this is the case, the chosen ACL implicitly constrains the performatives that should be expressed in $\mathcal{PT}_{\mathcal{ALC}}$. Moreover, ACLs usually allow an ontology to be explicitly referred to, so to allow the correct interpretation of the contents of a message. The ontology used by an overhearer is his own, and may or may not match (e.g., subsume) the ontologies used by services in their conversations (see the experiments in Section 5.1). It should be noted that nothing prevents an application from deploying multiple overhearers, with suggesters submitting $\mathcal{PT}_{\mathcal{ALC}}$ queries to those overhearers whose ontologies match theirs. The analysis of such a scenario, as well as optimizations of the interpretation of message content based on subsumption of overhearer's and services' ontologies, is left to future work.

An alternative feasible approach to the overhearer-suggester interaction is the adoption of temporal databases [8]. In this setting, the overhearer would manage an internal database updated with values extracted from the messages traveling on the channel, and the suggester would ask for notification by directly expressing temporal SQL queries (TSQL2, [19]). Some advantages can probably be gained from the point of view of performance and engineering, in particular if an application needs to permanently store the messages. On the other hand, since there is no ontological interpretation, the precision of the overhearing would be greatly reduced, affecting the quality of the suggestions. The solution with $\mathcal{PT}_{\mathcal{ALC}}$ is at a higher abstraction level and seems to be more general.

In the field of description logics, some systems are closely related to the one presented here (see [22] for an example, while [4] is an overview of temporal description logics). Often these formalisms are concerned with the modeling of concepts related to time *within* the terminological axioms, while in the overhearing architecture we are actually interested in the evolution in time of terminological expressions enriched with performatives. The latter does not mean that one cannot express temporal concepts in the ontology of the overhearer, but it does mean that these concepts do not have a specific temporal semantics assigned (e.g., like those achievable by using concrete domains [6]). Temporal concepts follow the semantics of the terminological language just like any other expressible concept.

7 Conclusions and Future Work

The work presented here originated in the field of cooperative software agents, and in particular from the overhearing architecture described in [10]. Our general focus is on interaction languages of a high abstraction level. In this work, we combined elements from temporal and description logic to provide a terse language for querying a log of messages exchanged among agents. In our opinion, its application within the overhearing architecture brings an useful enhancement to the state-of-the-art in agent collaboration, since it enables an easy detection of certain communication patterns; this may eventually lead to unsolicited help by collaborative agents.

In the current setting, the overhearer has the capability to tap a channel, to use an ontology and to parse query messages in $\mathcal{PT}_{\mathcal{ALC}}$. Mentalistic interpretation of conversations – that is, building a model of the behavior of the service agents – is entirely left to the suggesters. From a performance perspective, the overhearer acts as a filter that reduces the (potentially onerous) workload of the suggesters.

In Section 5, we mentioned a few ways of enhancing the current implementation of the overhearer. We are also considering giving him some simple mental recognition capability. If, for example, an agent is asking whether someone has a pen, one is willing to think that he does not have one and it is his intention to borrow a pen and then to use it. If an agent is asking about the result of a given soccer match, one is willing to think that he does not know about the result. Once the overhearer has this extra information, the suggester must be given the chance to make queries also regarding such content. We are thinking along the lines of epistemic logics [15]. Of particular interest is the temporal epistemic multi-agent logics presented in [12], which could be an interesting starting point. It would be necessary to work out a semantics in the same style of $\mathcal{PT}_{\mathcal{ALC}}$, where instead of the usual valuation function, a truth definition based on description logics reasoning mechanisms is used.

In addition to improving $\mathcal{PT}_{\mathcal{ALC}}$, future work on the overhearing architecture will look at the many conceptual and computational challenges involved in understanding when to intervene in a conversation, and how to deal with unsolicited suggestions.

References

1. *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org`.
2. *JACK Intelligent Agents ver. 3.0, User Manual*. Agent Oriented Software, 2001. `http://www.jackagents.com`.
3. *JavaCC Documentation, ver. 2.0*. Metamata and Sun Microsystems, 2001. `http://www.metamata.com/javacc/`.
4. A. Artale and E. Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 2001. To appear.
5. F. Baader, H. Burckert, J. Heinsohn, B. Hollunder, J. Muller, B. Nebel, W. Nutt, and H. Profitlich. Terminological knowledge representation: a proposal for a terminological logic. Technical report, DFKI, Saarbrucken, 1992.
6. F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *IJCAI*, pages 452–457, 1991.
7. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
8. M. Boehlen, J. Chomicki, R. Snodgrass, and D. Toman. Querying TSQL2 databases with temporal logic. In *5th International Conference on Extending Database Technology (EDBT)*, Avignon, 1996.
9. A. Borgida and P. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. *JAIR*, 1:277–308, 1994.
10. P. Busetta, L. Serafini, D. Singh, and F. Zini. Extending Multi-Agent Cooperation by Overhearing. Technical Report 0101-01, ITC-irst, Trento, 2001.
11. P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In *Proceedings of the AAAI Spring Simposium on Software Agents*, pages 1–8. AAAI Press, 1994.
12. C. Dixon, M. Fisher, and M. Wooldridge. Resolution for Temporal Logics of Knowledge. *Journal of Logic and Computation*, 8(3):345–372, 1998.
13. J. Doran, S. Franklin, N. Jennings, and T. Norman. On Cooperation in Multi-Agent Systems. *The Knowledge Engineering Review*, 12(3), 1997.
14. T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.
15. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
16. I. Horrocks. Using an expressive description logic: FaCT or fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647. Morgan Kaufmann Publishers, San Francisco, California, 1998.
17. M. Klusch, editor. *Intelligent Information Systems*. Springer-Verlag, 1999.
18. T. Oates, M. Prasad, and V. Lesser. Cooperative Information Gathering: A Distributed Problem Solving Approach. *IEEE Proc. on Software Engineering*, 144(1), 1997.
19. B. Salzberg and V. J. Tsotras. Comparison of Access Methods for Time-Evolving Data. *Computing Surveys*, 31(2):158–221, 1999.
20. M. Schmidt-Schau and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.
21. Y. Venema. Temporal logics. In L. Goble, editor, *Blackwell's Guide to Philosophical Logic*. Basil Blackwell Publishers, Cambridge, MA, To appear.
22. F. Wolter and M. Zakharyaschev. Temporalizing description logics. In *FroCoS'98*, Amsterdam, 1998. Kluwer.