1. Introduction

1.1. Context

Artificial assistant research is conducted in both industrial and academic projects to propose hardware and software solutions that efficiently support users in a wide range of activities [1, 2, 3].

The concept of software agents has been increasingly applied to the case of personal assistants, so that the two terms are now unified. In this context, an agent is a software endowed with some extent of autonomy and interaction capabilities. The potential success of this technology is indeed due to the natural mapping between agent and user. Such a duality allows both human user and system developer to better deal with the software as an embodied entity at runtime and design time.

New applications are under development with this technology, and the idea of ‘intelligent assistant agent’ is already well-spread with a certain variety of results to demonstrate [1, 2, 3, 4, 5, 6].
The second case devotes a set of agents to a human team and so enters the paradigm of Multi-Agent Systems (MAS). For instances, the M project supports a virtual meeting room metaphor to let users work together and remain synchronised while in different physical places [3]. The ELECTRIC ELVES system is a team of agents devoted to manage the schedule of a user group [2]. It handles events to maintain the global coherence of the schedule (real-time reorganisation, etc.) and can also book real facilities such as restaurants. Instead of working in isolation, agents in MAS can be designed to cooperate, help or compete each other. Therefore, the ‘intelligence’ of each agent can be completed by interacting with others and exploiting their services. However, the MAS approach was so far applied to support user groups with global assistants. As before, there is no assignment of more than one private assistant per user, and our proposal is to extend individual assistance with teams.

In teams, interactions (in our case communications) become as important as agents to leverage a common activity. The current view on interactions is however limited to direct links between agents. Consequently, agents are not sensible to a whole class of events that occur indirectly in teamwork, such as behaviour recognition (conflicts, deadlocks, implicit call for help, etc.). In this work, we exploit the concept of ‘overhearing’ to model indirect interactions. It was successfully applied in related project such as group formation [8], conversation recognition [9], or communication [7], but the introduction in the interaction infrastructure of MAS is not addressed yet.

1.4. Toward Interactive Services

In this paper, we propose a methodology to describe the infrastructure of assistant teams with overhearing. In Section 2, our approach on assistant teams is first presented in further details, and the ‘T-compound’ is introduced as a model of indirect interactions for overhearing. Section 3 deals with the core of our work on the interactive infrastructure for the team, which is an abstract methodology to describe MAS that exploit both traditional interactions and overhearing. Section 4 details an example to design interactions following two instances of our methodology. Finally, Section 5 discusses this paper and Section 6 summarizes it before presenting future research.

2. Assistant Teams and Overhearing

2.1. Assistant Teams

In this paper, we support the idea of a team of specialised cooperative agents to assist one user as shown on Fig.1, where two users interact through computer-based systems (instant messaging, email, etc.).

![Figure 1. Assistant Team Architecture](image)

The left user is supported by a team of assistants that can reply to individual requests and take initiatives. Hence, this team should validate the two features identified in section 1.2. In addition, assistants can perform a teamwork as a MAS for their common user. Agents can support each other, so that their cooperative work can lead to more advanced results by composition. For instance, a travel agent and a schedule agent can collaborate to find autonomously better answers to a request from the user.

This architecture allows reducing the complexity of each agent, since none need to have comprehensive capabilities and may rely on others. Maintenance is also improved, since faulty agents can be revised or removed from the system population. Moreover, the two features of assistants presented in section 1.2 can rely on a larger range of agent abilities. For example, ‘legacy’ agents (such as [1, 4, 5, 6]) extended with team capabilities could still answer requests related to their specialities and they could also interact to get information, delegate tasks, or ask for help. Agents become service providers for each other, on behalf of the user.

Assistant teams require two endeavours in our approach, due to a separation of concerns between agent interactions that define the infrastructure of the system and contents that transit according to service definitions. The infrastructure describes how agents can work together to perform services provided by the application. The service definition details agent architectures and capabilities to define what they do as service providers. The next sections describe our methodology to deal with the interaction layer, while the service definition shall be the concern of application-dependent work.

2.2. Extending the Interaction Paradigm with Overhearing and the T-compound

Fig.1 particularly emphasizes the interaction aspect of assistant teams. In addition to the request-reply loop and initiatives, teamwork takes advantage of the variety of in-
interaction means among agents and users to foster their cooperation. Beyond direct interactions, we introduce ‘overhearing’ [7, 8, 9] to expand the initiative and teamwork potentials of assistants. Overhearing occurs when two agents are interacting, say discussing, and a third agent can hear their conversation as shown on Fig.2.

![Figure 2. Concept of Overhearing](image)

Overhearing lets agents listen to more information than present computer solutions provide and extract further relevant clues to support the user. This situation is frequent in our daily life, but is not exploited in the computer world, due to the point-to-point nature of electronic communications. This concept was shown to be meaningful in the cooperation case [7, 8, 9], where agent reactivity and capabilities are expanded. However, overhearing also suffers from two main issues. First, it can be thought of as eavesdropping, especially in self-interested agent systems. Second, the flow of interactions can become overwhelming owing to the multiplication of sources, so that message selection and filtering algorithms would be required. In this paper, we focus on infrastructures for the cooperation case and do not deal with these issues.

Fig.2 depicts our model of overhearing named T-compound to capture as one entity the interactive schema of overhearing. The aim is to design MAS based on three building blocks, namely agents (A), traditional point-to-point interactions (I), and indirect interactions such as overhearing (T). We think this new block is general enough to deal with assistant teams as well as other types of MAS.

MAS frameworks and methodologies deal with interactions mainly through the definition of protocols [14, 15]. By introducing the ‘T’, interaction protocols are still the main stream to specify the behaviours of agents, but current notations can sometimes lack clarity. Overhearing enables the use of indirect sources by agents and a formalism such as AUML [16] was not tailored to clearly differentiate them from direct sources. This distinction is important for system designers to see and control overhearing cases. Special care is required to avoid potential breaches or ‘side-effects’ that could lead to eavesdropping. The remainder of this paper details a methodology we developed to build supplementary views on system interactions to support designers in the management of overhearing.

3. Modelling the Interactive Infrastructure

3.1. Background

The methodology we propose hereafter was originally developed along the T-compound [11]. The purpose was to treat homogeneously direct interactions and overhearing in MAS. So far, MAS frameworks handle the bricks A and I introduced in 2.2. Our methodology extends them with T.

Instead of interaction protocols, this methodology produces two views on MAS interactions, namely $\mathcal{I}_I$ and $\mathcal{I}_A$. $\mathcal{I}_S$ represents interactions at the system level. It contains the integrality of I and T interactions in the system. On the other hand, $\mathcal{I}_A$ is a representation from the agent local view, instead of the global one from the system. Interactions in the system are grouped by agents, so that each group summarizes all interactions an agent can deal with in the system.

Hereafter, we describe an abstract methodology to build the two views $\mathcal{I}_S$ and $\mathcal{I}_A$, and we illustrate it with two implementation instances on one example.

3.2. Methodology

3.2.1. Initial Conditions. At the beginning of this process, we consider the interaction specifications of the target MAS as a set $I$ to build the two sets $\mathcal{I}_S$ and $\mathcal{I}_A$. Initially, $I$ contains all specified interactions with possible redundancies and inconsistencies. $\mathcal{I}_S$ and $\mathcal{I}_A$ are empty.

3.2.2. Process sequence. The following procedure is reduced to abstract functions that specify what instances of methodologies need to implement.

1. $I$ is a set of interactions, $\mathcal{I}_S = \emptyset, \mathcal{I}_A = \emptyset$
2. \textbf{process_redundancies($I$)}
   \textbf{Parameter} Interaction set $I$ (input/output)
   \textbf{Description} Remove all redundancies from $I$.
3. \textbf{process_combinations($I$)}
   \textbf{Parameter} Interaction set $I$ (input/output)
   \textbf{Description} Compose interactions according to compatibility rules.
4. \textbf{minimise($I$)}
   \textbf{Parameter} Interaction set $I$ (input/output)
   \textbf{Description} Remove interactions from $I$ according to precedence rules.
5. $\mathcal{I}_S = I$
6. \textbf{rewrite($\mathcal{I}_S, \mathcal{I}_A$)}
   \textbf{Parameters} $\mathcal{I}_S$ (input), $\mathcal{I}_A$ (input/output)
   \textbf{Description} Identify interaction actors and compile a group of interactions per agents.
First, process_redundancies cleans the interaction set \( I \) from redundant elements. For instance, two protocols may define two distinct discussions between A and B as ‘conversation(A,B)’. However, these specifications only need one discussion link with respect to the interaction infrastructure, since only the contents differ. Second, process_combinations follows rules to combine compatible interaction types. Typically, the specification may feature separately ‘talk(A,B)’ and ‘talk(B,A)’. As these interactions rely on the same agents, we consider they are compatible and should be composed as ‘conversation(A,B)’. This interaction has an equivalent infrastructure to the two individual talks; it has a clearer semantics and is more compact. Third, minimise exploits precedence rules to eliminate weaker interactions that do not add any information. For example, ‘talk(A,B)’ is weaker than ‘conversation(A,B)’ in terms of infrastructure: the former is ‘included’ in the latter, and so can be removed. Finally, \( I_S \) is equal to the result of the ordered sequence of procedures and rewrite produces \( I_A \) from \( I_S \). This last step browses \( I_S \) and yields all individual agents with the interactions they participate in. If \( I_S \) ends with \{'conversation(A,B)', 'conversation(B,C)'</\> then \( I_A \) contains the three agents A, B, C with their respective interactions: A and C participate in a conversation, B in two.

The following Section 4 illustrates this abstract process in two cases of formula- and pattern-based approaches. The principles are the same since we restrict the steps to the handling of interaction artefacts and their assembly up to the aimed MAS. However, the representation is different: the first method relies on formula rewriting, and the second is based on graphical design.

4. Methodology through an Example

4.1. Illustrative Scenario

In this scenario John and Takezo have decided to organise a meeting, and they agree on the details by email. John uses an ordinary mailer, whereas Takezo also has an assistant team populated as follows:

- **Presentation Agent** handles the user interface, i.e. all events from and to the user.
- **Search Agent** fetches information for the user as a search engine. It also initiates search to find and preload relevant data in the user current context.
- **Schedule Agent** handles the user schedule. It also points out inconsistencies and asks for specific search (timetable, etc.) to the search agent.

The purpose of this section is to represent the interactions required for this system. The interest of overhearing in this case arises in frequent situations. For example, let us consider a use case.

Takezo checks his availabilities (through the Presentation Agent) with the Schedule Agent and sends a message to John with a proposal. While Takezo interacted with the Schedule Agent, the Search Agent was idle and could overhear the ‘discussion’ and initiate a train time table search. It realises there are work engineering on the potential line the user may choose, so it informs Takezo about this issue.

In this example and from the system specification, we deduce that every pair of agents need interact directly. In addition, the ability of the Search Agent to overhear adds one T-compound in this use case. These interactions let agents work together, and authorise Search Agent to react more autonomously to other agents’ behaviours.

4.2. Interaction Models

4.2.1. Formula-Based Modelling. Our formal model of overhearing introduces a set of compounds to cover the different interaction cases uniformly [11]. Fig.3 and 4 lay out the five situations we distinguished, and we explain thereafter their semantics.

\[
\begin{align*}
M(A,B) & \quad \text{Use } \alpha. \\
D(A,B) & \quad \text{Use } \alpha \text{ and } \beta. \\
T_0(A,B,P) & \quad \text{Use } \alpha \text{ and } h_\alpha. \\
T_1(A,B,P) & \quad \text{Use } \alpha, \beta \text{ and } h_\alpha. \\
T_2(A,B,P) & \quad \text{Use } \alpha, \beta, h_\alpha \text{ and } h_\beta.
\end{align*}
\]

**Figure 3. Patterns with 2 agents**

The ‘monologue’ M represents the sending of messages from one agent to another, without possible answer (master/slave, dictatorial management, etc.). The ‘duplex’ D completes M with the answer link, and so depicts a conversation.

**Figure 4. Patterns with 3 agents**
$T_0$ appears as soon as the sending agent A informs another agent P about messages it sends to a peer B. There is no conversation yet between A and B. In $T_1$, A and B converse and P overhears only A. This situation is necessary for completeness. Typically, this model represents a scene where A and P are together without B, and A calls B on the phone. In such a situation, P can only hear the talk of A. Finally, $T_2$ extends $T_1$ to let P also overhear B.

Regarding the example scenario, the instance of our methodology leads to handling the above compounds for building the interaction views. The process based on the specifications yields the following, where each procedure algorithm is detailed in [11].

1. $I = \{D_1=D($SearchA,PA$), D_2=D($SearchA,ScheduleA$), D_3=D($PA,ScheduleA$), T_2=T_2($PA,ScheduleA,SearchA$)\}$, $I_S = \emptyset$, $I_A = \emptyset$
2. $\text{process\_redundancies}(I) = I$
3. $\text{process\_combinations}(I) = \{M_1=M($SearchA,PA$), M_2=M($SearchA,ScheduleA$), T_2\}$
4. $\text{minimise}(I) = \{M_1, M_2, T_2\}$
5. $I_S = \{M_1, M_2, T_2\}$
6. $\text{rewrite}(I_S, I_A)$ is detailed hereafter.

The procedure first searches for redundancies (line 2) and, as none is found, $I$ remains unmodified. Then, combinations are performed (line 3) to optimise the set of interaction. $I$ features three duplex D and one T-compound $T_2$. The duplex are indeed partially included in the $T_2$, so that this stage rewrites the set. $D_3$ is already in $T_2$ so it is eliminated. $D_1$ and $D_2$ have one of their directions included in $T_2$ —one monologue is already supported— so that they are respectively replaced by the missing direction, i.e. $M_1$ and $M_2$. The minimisation process (line 4) is aimed to filter out irrelevant interactions. In this example, $M_1$, $M_2$ and $T_2$ are needed and not related so that all are kept. In the end, the procedure produces $I_S = \{M_1, M_2, T_2\}$.

The last stage is to compile $I_A$. The process relies on the underlying syntax of the formal model [11] and decomposes $T_2$ into $D_4=D($ScheduleA,PA$)$ for the overheard discussion, and two unidirectional channels $M_3=M($PA,SearchA$)$ and $M_4=M($ScheduleA,SearchA$)$ to carry the overheard messages. Thus, we have:

$I_A=\{I_1, I_2, I_3\}$ where

$I_1=\{M_1, M_3, D_4\}$ for PA
$I_2=\{M_2, M_4, D_4\}$ for ScheduleA
$I_3=\{M_2, M_3, M_4\}$ for SearchA

For instance, $I_A$ shows that PA participates in three interactions in the system. The type of these interactions describes part of the roles and capabilities of PA. It can receive messages from SearchA ($M_1$), reply ($M_2$), and converse with ScheduleA ($D_4$), as specified initially.

### 4.2.2. Design Pattern Approach

Instead of modeling the system using formulae, the design pattern approach is graphical and based on the correspondence between schemas and formulae of the previous part.

In object-oriented software engineering, the analysis of the target system can end with the selection of design patterns to build up the system [13], if this method is appropriate. We extended this principle to agents considering the T-compound as an agent-specific design pattern. Fig.3 and 4 show the complete set of patterns to ensure an homogeneous coverage of possible interactions, ranging from unidirectional speech act to the T-compound.

The method is the same as before, though notations now refer to graphical patterns instead of formulae. We emphasize hereafter the two stages that produce $I_S$ and $I_A$, i.e. ‘process\_combinations’ and ‘rewrite’ as in section 4.2.1. Fig.5 shows one of the combination procedures, and Fig.6 illustrates the construction of $I_A$.

**Figure 5. Weaving Interaction Compounds**

**Figure 6. Visualisation of $I_A$**

From the specification of $I$, the procedure combines $T_2$ and $D_1$ since these two interactions are compatible. In terms of patterns, we reuse the terminology of ‘weaving’ [13]. On Fig.5, this weaving completes $T_2$ with the $M_1$ interaction from SearchA to PA. Indeed, this $M_1$ is the part of $D_1$ that contribute to the interaction infrastructure defined by $T_2$. The remaining part of $D_1$ is redundant with $T_2$, so it can be eliminated. In the example, the same is performed for $D_2$ to end with $I_S = \{M_1, M_2, T_2\}$.

Fig.6 shows the complete pattern weaving for the system, i.e. the set $I_S$. From this image, $I_A$ is deduced by identifying the surrounding of each agent. For example, ScheduleA can engage in three interactions depicted as the three
arrows connected to its representation. ScheduleA can receive and send messages with respect to PA and SearchA (with the overhearing constraint). The same is done for SearchA and PA to yield $I_A$.

5. Discussion

Our approach of assistance distinguishes two concerns in the construction of assistant teams, namely the infrastructure and service definition. The methodology we propose in this paper addresses the former concern and produces two views on interactions: $I_S$ and $I_A$. From these two sets, system designers respectively obtain a model of all interactions including overhearing and the list of interactions per agent. The former can be useful to constrain the system (e.g. prevent some overhearings), and the latter leads to the definition of roles in interaction protocols as complement view to the lifeline in AUML [16].

Considering the purpose of assistant teams, these two sets only describe statically the system in the present state. Indeed, our process is devoted to the design stage and must be applied before the system launch. For the run-time, another modeling is required to take into account the dynamic creation and removal of interaction patterns. Consequently, the agent population remains constant with the current method, which is a strong constraint in open MAS. In the case of our assistant teams, it means the user can neither configure the team composition, add nor remove service providers.

6. Conclusion

Personal assistant teams appear in this paper as an extension of present personal assistants. This technology improves the conceptual framework to design more flexible systems populated with potentially less complex interactive agents.

In order to foster cooperation among assistants, we introduced in MAS the notion of overhearing through the T-compound. This interaction pattern allows developing the interaction potential of agents so that they can be more reactive and take into account new classes of events, such as group behaviour recognition.

The present assistant teams interact only with their owners, and future stages should allow them to deal with other teams and users to be fully coherent with present endeavours in MAS. Besides, the work performed in Negotiation Support Systems evolves in that direction as well. The current methodology also needs to be revised to become dynamic. It is currently static and cannot fit the requirements for open systems. This work may eventually lead to a larger framework, since computing facilities deeply enter our daily-life. Future scenarii exploiting assistant teams could even consist in agent families devoted to manage our ‘digital life’.

References


