Tutoriel : Open Agent Architecture Développement d'applications de TALN distribuées, multiagents et multiplates-formes

Antonio BALVET (1), Olivier GRISVARD (2) et Pascal BISSON (2)

(1) UMR MoDyCo, Université Paris X Nanterre
200, av. de le République 92001 Nanterre
antonio.balvet@u-paris10.fr
(2) Thales RT, DAS-HIT
Domaine de Corbeville, 91404 Orsay
olivier.grisvard@thalesgroup.com
pascal.bisson@thalesgroup.com

Résumé - Abstract

Nous présenterons tout d'abord la philosophie « Agents » en général, afin d'en montrer les avantages pour le domaine du TALN, qui se caractérise par une hétérogénéité avérée des systèmes existants (multiplicité des langages de programmation), ainsi qu'une forte demande en ressources (mémoire notamment). Nous ferons ensuite une présentation des principales plate-formes orientées agents, puis nous examinerons de plus près la plate-forme développée au Standford Research Institute (SRI) : OAA (licence libre). Nous clôturerons le tutoriel sur des exemples commentés d'applications industrielles utilisant OAA, permettant de donner toutes les clés nécessaires au développement d'applications distribuées (intra/internet), multiagents et multiplates-formes (plusieurs langages de programmation/systèmes d'exploitation).

We will first present the "Agent" philosophy in general, in order to put forward its usefulness for resources-intensive (memory) NLP systems, in a heterogeneous context. We will also present an overview of the main agent-oriented platforms, with a focus on the OAA platform, developed at the Stanford Research Institute. Finally, we will present a detailed example of a real-scale multiagent NLP system based on OAA, with an emphasis on development details which will provide the audience with all the required information to develop distributed (intra/internet), multiagent and multiplaforms systems (different programming languages/operating systems).

Introduction

La tendance actuelle, dans le domaine du Traitement Automatique du Langage Naturel, est à la standardisation, la modularité et la réutilisabilité, après une période de foisonnement dans les approches, les langages de programmation utilisés ainsi que les plates-formes visées. Cette tendance semble suivre le modèle du domaine du développement logiciel professionnel, elle se traduit également depuis une dizaine d'années par des campagnes d'évaluation internationales portant sur des domaines traditionnels du TALN : étiquetage, levée d'ambiguïtés, calcul sémantique etc… (les conférences MUC, TREC, mais également AMARYLLIS, Senseval ou Parseval), ainsi que la constitution de corpus de référence.

La nécessité de disposer d'environnements de développement, de boîtes à outils et autres plate-formes normalisées pour le TALN se fait d'autant plus sentir qu'un certain nombre de techniques sont devenues, de fait, des standard dans le domaine (ex : l'analyse locale, les cascades de transducteurs) et que, par ailleurs, un certain nombre de ressources lexicales sont disponibles pour les langues autres que l'anglais (ex : EuroWordnet, Memodata). Toutefois, force est de constater l'hétérogénéité résiduelle du domaine, au niveau logiciel, notamment en ce qui concerne les langages de programmation utilisés (de Lisp à Perl en passant par C++, Java et Prolog), le mode de développement (programmation structurée/orientée-objet/orientée-composants/multiagents) que le cadre de déploiement (ex : sur poste local / sur un réseau via une architecture client-server, sous Windows/Linux/Unix).

Certaines initiatives, telles la plate-forme GATE (General Architecture for Text Engineering), développée à l'université de Sheffield, ont tenté de répondre à cette demande (voir Cunningham H. *et alii.*, 1997) en proposant une architecture modulaire visant à libérer les linguistes-informaticiens de tâches de développement inutiles, répétées pour chaque implémentation, telles que la gestion des documents, la visualisation des données, les liaisons entre modules logiciels existants. Toutefois, malgré la popularité de GATE[1] pour des tâches liées au traitement de l'écrit (extraction d'information notamment), la plate-forme développée par l'université de Sheffield n'offre pas de fonctionnalités telles que : le fonctionnement distribué (sur un réseau intra/internet) permettant de répartir la charge des traitements sur des serveurs spécialisés en laissant l'affichage au(x) poste(s) client(s), ou encore le fonctionnement par agents autonomes communiquant par requêtes via un superviseur (architecture de type « tableau noir »). Par ailleurs, GATE ne s'aventure pas hors du domaine fixé par ses concepteurs (i.e. l'écrit), alors que l'intégration des différentes modalités du langage naturel (voix, écrit, ou encore geste) apparaît comme l'étape suivante dans les développements pour le domaine du TALN.

Nous présenterons au cours de ce tutoriel une architecture en libre accès, développée au SRI pour le développement de systèmes à dimension industrielle, hétérogènes, distribués, multi-agents et multi-plateformes : OAA (Open Agent Architecture). Cette architecture permet d'envisager le développement d'applications de TALN comme l'assemblage de briques de base, favorisant l'émergence de véritables « systèmes » dont l'intégration, le déploiement et

[1] Voir Maynard D. *et alii.,* 2000 : plus de 12 projets de dimension européenne et industrielle reposant sur GATE ont été recensés.

le fonctionnement reposent sur des bases solides, ainsi que la réutilisation et la capitalisation de modules logiciels existants[2].

Nous assoirons le présent tutoriel sur l'expérience acquise par le laboratoire Thales-RT France dans le domaine du développement d'applications intégrant la commande vocale, le traitement du texte et la réalité augmentée reposant sur OAA. Dans un premier temps, nous présenterons, de façon synthétique, la philosophie « Agents », en mettant l'accent sur les bénéfices que le domaine du TALN peut en tirer. Dans un deuxième temps, nous présenterons les spécificités de la plate-forme OAA par rapport aux principales plates-formes agents/distribuées disponibles. Dans un troisième temps, nous aborderons des exemples commentés de systèmes industriels de TALN, développés à Thales-RT France, reposant sur OAA ainsi que sur des composants aussi divers que Nuance (reconnaissance vocale) ou le Dictionnaire Intégral (Memodata).

Le présent tutoriel s'appuie principalement sur les références consultables sur le site du SRI, consacré à la plate-forme OAA, version 2.1 et à ses applications : http://www.ai.sri.com/~oaa. Sont notamment disponibles :

- Le guide du développeur (OAA 2.1 Developer's Guide) ;
- La documentation du système (OAA 2.1 Documentation) ;
- Un tutoriel sur OAA (OAA 2.1 Tutorial) ;
- La référence du langage ICL (OAA Interagent Communication Language, API Reference Manual) ;
- La Foire Aux Questions OAA 2.x (OAA V2.x FAQ) ;
- Le manuel de référence de la libraire Agent, fournissant l'accès aux fonctionnalités OAA pour l'intégration (OAA, Agent Library Reference Manual Version 2.1) ;
- L'API OAA pour les langages C et Java ;
- Des démonstrations vidéo d'applications OAA.

Par ailleurs, des exemples d'applications centrées sur l'interaction multimodale avec l'utilisateur sont présentées, ainsi que des exemples de code source pour l'ensemble des langages de programmation supportés : C, Java,, Prolog et WebL. L'intégralité du code source OAA est également disponible.

---

[2] OAA a été adoptée par de nombreux laboratoires à travers le monde, tant privés que publics.

1 La philosophie « Agents »

1.1 Aperçu de différents paradigmes de développement logiciel

La figure ci-dessous présente de façon synthétique les principaux modèles de développement de logiciels.

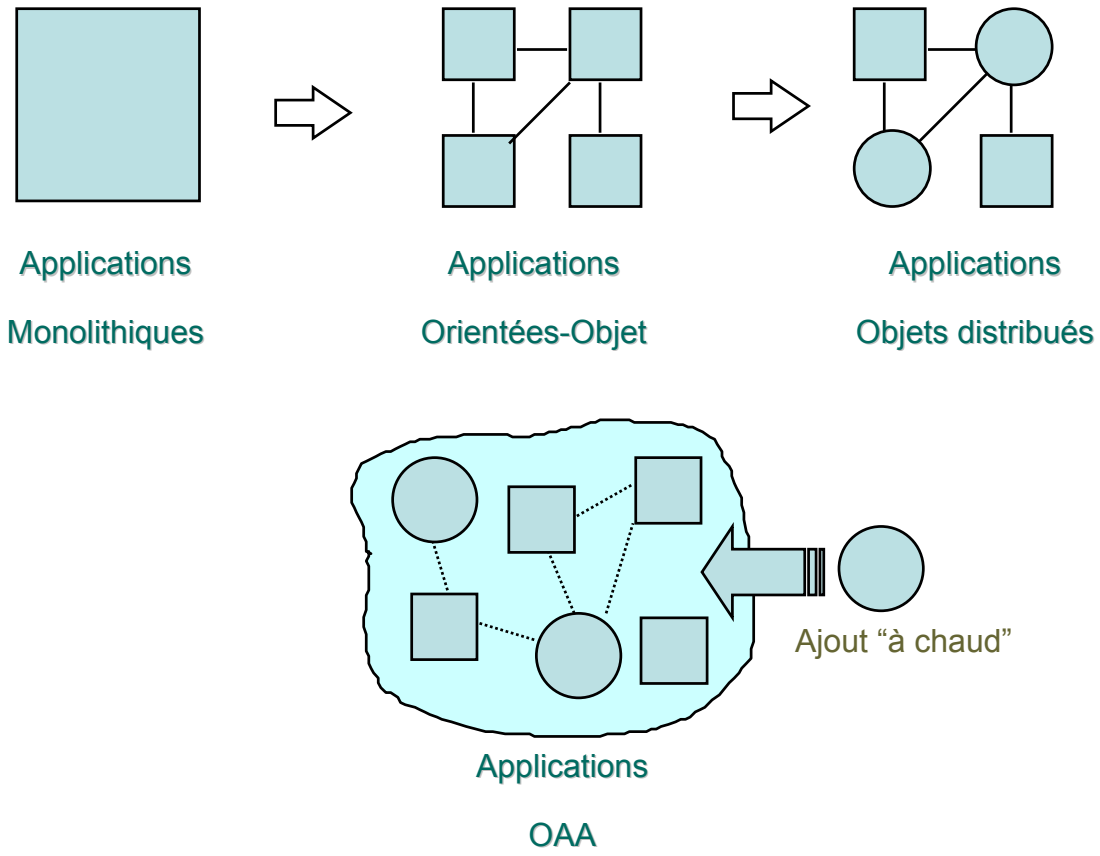

Figure 1 : principaux modèles de développement de logiciels

Dans cette figure, nous cherchons à souligner le passage d'applications « monolithiques », constituant un tout indivisible, dont les différentes fonctionnalités ne pouvaient être dissociées, aux applications « orientées-objets », puis « objets distribués ». Le passage aux applications orientées-objets ont essentiellement apporté au monde du développement la modularisation, ainsi que la possibilité de faire hériter des fonctionnalités intéressantes à de nouveaux composants. Cette étape a surtout permis à la communauté des développeurs de pouvoir envisager la réutilisabilité des composants logiciels autrement que par le code source. Le paradigme distribué a, lui, permis le développement d'applications non centralisées sur un poste unique : le fonctionnement en client/serveur. L'adoption de ce modèle a également permis d'envisager le développement d'architectures « n-tier » (à n étages), permettant, par

exemple, de limiter la charge des postes clients à l'affichage (interface-utilisateur), en déléguant les processus gourmands en ressources à des postes spécialisés (serveurs).

OAA est présenté par ses concepteurs comme l'étape suivante dans les modèles de développement : les composants logiciels modularisés forment une communauté, dont les interactions sont réglées par un composant central, le facilitateur. Dans cette optique, les composants peuvent être ajoutés ou retirés de la communauté en fonction des besoins, c'est-à-dire en fonction de la tâche, des ressources disponibles etc… Cet ajout ou retrait se font « à chaud » : chaque nouvel agent annonce au facilitateur les services qu'il est capable de fournir.

1.2 Les bases des plates-formes multiagents

1.2.1 Modularité

La modularité semble être une des caractéristiques principales des architectures agents. Cette modularité permet d'envisager des « systèmes » logiciels, dans lesquels des briques de base sont assemblées de façon à fournir un service global. Cette approche du développement logiciel permet également d'envisager des systèmes où les composants sont interchangeables.

La modularité, en termes logiciels, présuppose d'adopter des modèles ou principes d'architecture où les services sont distingués en fonction de leur type. Ainsi, par exemple, le principe « modèle/vue/contrôleur » oblige à distinguer les différents objets logiciels selon qu'ils offrent des services d'affichage (interface-utilisateur), de contrôle de l'application (prise en charge matérielle, suivi du déroulement des programmes) ou de représentation abstraite (ex : pour un document XML, la DTD permettant d'interpréter le balisage employé).

1.2.2 Autonomie

Au-delà des caractéristiques évoquées plus haut, la philosophie agents repose sur la notion d'autonomie. En d'autres termes, les agents logiciels sont vus comme des extensions des utilisateurs : ils sont pensés pour accomplir des tâches à la place des utilisateurs. Ceci suppose que lesdits utilisateurs spécifient de façon formelle les tâches à accomplir, ainsi que les modalités de l'exécution de ces tâches : synchronisation temporelle, environnement, conditions d'exécution … En d'autres termes, l'autonomie accordée aux agents logiciels présuppose une caractérisation explicite formelle des buts poursuivis par les utilisateurs, que les agents vont tenter de mener à bien en fonction des conditions du monde extérieur (environnement logiciel). Cette autonomie implique également une représentation logique de l'agent, de son environnement et des relations qu'il entretient avec cet environnement. C'est pourquoi les formalismes d'inspiration logique et la programmation déclarative, tels que fournis par le langage Prolog, par exemple, servent souvent de base aux architectures agents.

1.2.3 Fonctionnement dynamique

Dans le cas de OAA, l'essentiel du comportement des agents regroupés en communauté autour d'un facilitateur est géré par une couche de représentation logique des individus composant la communauté, de leurs buts, des services qu'ils peuvent rendre. Cette couche

logique repose sur un moteur Prolog, qui permet, par le biais du principe d'unification, de mener à bien des buts de façon non procédurale, en optimisant l'accomplissement d'une tâche par rapport aux agents et services disponibles.

Le fonctionnement dynamique permet l'ajout ou le retrait d'agents de la communauté sans arrêt du système, ni recompilation. Le facilitateur enregistre simplement un changement dans la communauté qu'il régule.

1.3 Exemple : l'assistant bureautique

La figure ci-dessous donne un aperçu du prototype « office assistant » développé par le SRI.

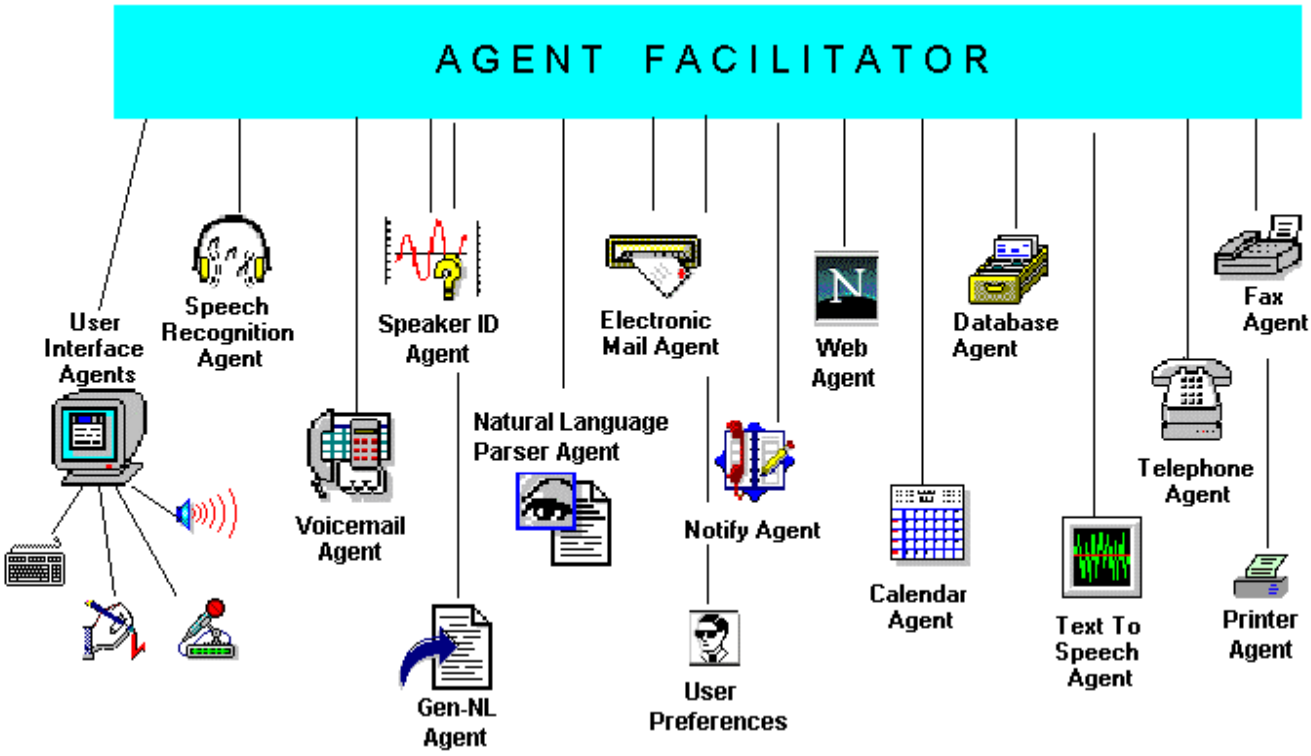

Figure 2 : communauté d'agents formant le « office assistant »

Ce prototype intègre un ensemble d'agents logiciels bureautiques, en interaction constante avec les utilisateurs (reconnaissance vocale, reconnaissance de locuteur, prise en compte des préférences-utilisateurs, prise en charge des périphériques : fax, e-mail …). L'ensemble de ces agents forme une « communauté », dont les interactions sont régies par le facilitateur. A tout moment, de nouveaux agents, prenant en charge d'autres périphériques, peuvent être ajoutés.

2 OAA et autres plates-formes multiagents

Dans cette partie, nous présentons synthétiquement les spécificités de quelques plates-formes multiagents reconnues.

2.1 Quelques plates-formes multiagents

2.1.1 Principes communs

D'après les concepteurs d'OAA, la plupart des plates-formes multiagents visent à assurer un fonctionnement modulaire, autonome et distribué de leurs agents. Ceci passe par la mise en œuvre d'au moins quatre composants :

- un protocole de communication assurant l'acheminement asynchrone des messages ;
- un protocole d'interaction définissant différentes modalités de communication ainsi que leurs implications « sociales » (ex : une requête appelle une réponse) ;
- un langage de contenu (une sémantique) permettant l'expression et la compréhension d'énoncés ;
- une ontologie, ou ensemble partagé d'associations entre concepts et interprétation, ainsi qu'un vocabulaire commun.

2.1.2 Quelques plates-formes agents

Historiquement, l'une des plates-formes les plus reconnues et les plus diffusées semble être KQML (Knowledge Query and Manipulation Language)[3]. KQML, qui gère les aspects liés au protocole d'interaction, est souvent associée à KIF (Knowledge Interchange Format)[4], pour la gestion du contenu et la mise en place d'ontologies génériques aussi bien que spécialisées. KQML inaugura l'utilisation de « performatifs symboliques » afin de gérer la notion de but dans les interactions. Cette plate-forme, plutôt orientée vers le traitement automatisé des interactions vocales, apparaît limitée par l'utilisation d'un ensemble fini de performatifs atomiques, limitant le pouvoir expressif du formalisme.

BDI (Belief, Desire and Intention), une autre approche reconnue, fait des hypothèses plus fortes quant aux connaissances et aux processus mis en œuvre par les différents agents. Cette approche repose sur la structuration des activités de ses agents autour des concepts de croyances, de désir et d'intentionnalité (Rao & Georgeff, 1995). Bien que BDI ait contribué à intégrer un fort niveau d'abstraction dans les plates-formes multiagents, les applications sont apparues limitées par les présupposés structuraux liés à chaque agent, mais également par dans le cadre de la pérennisation d'applications (legacy code).

2.2 La philosphie OAA

Les concepteurs de la plate-forme revendiquent une parenté forte avec les approches mentionnées plus haut. Toutefois, ils mettent en avant la pérennisation d'applications,

---

[3] Voir (Labrou & Finin, 1997) et (Finin *et al.*, 1997).

[4] Voir (Geneserth & Fikes, 1992).

l'intégration de composants hétérogènes, un fonctionnement dynamique et extensible proche des approches dites « tableau noir » (blackboard), ainsi que l'efficacité des agents dits mobiles et les interactions de haut niveau des agents communicationnels. Par ailleurs, les concepteurs inscrivent d'emblée leur plate-forme dans le cadre des interactions multimodales intelligentes avec les utilisateurs, ainsi que dans celui des systèmes robustes à vocation industrielle.

2.2.1 Des communications de haut niveau : ICL

Le protocole de communication inter-agents, Interagent Communication Language, est la *lingua franca* de la communauté d'agents gérée par un facilitateur. Elle repose sur le formalisme Prolog (programmation déclarative par expression de buts à atteindre et de conditions pour réaliser ces buts). Elle permet une abstraction par rapport à l'ensemble des détails propres à chaque plate-forme.

ICL intègre une couche de protocole de communication similaire à celle fournie par KQML, ainsi qu'une couche de contenu, semblable à celle de KIF. La couche de communication d'ICL est définie par les types d'événements, ainsi que les listes de paramètres associés à certains de ces événements. La couche d'événements, elle, regroupe les buts spécifiques, les *triggers* (déclencheurs d'événements) et les données pouvant être accessibles par les différents événements.

Les concepteurs d'OAA recommandent, dans la mesure du possible, d'avoir recours à ICL, bien qu'il soit possible de gérer des événements, des données etc… de façon classique pour chaque composant (dépendante du langage de programmation). Les concepteurs mettent en avant les avantages à passer par ICL : les différents buts et sous-buts sont rendus accessibles au facilitateur, qui peut ainsi répartir la charge sur les différents agents. Un meilleur contrôle des requêtes peut ainsi être assuré par le facilitateur.

2.2.2 Notion de « délégation »

Cette notion est centrale dans l'approche OAA : de façon générale, les requêtes, quel que soit leur type, ne sont pas adressées directement par le demandeur au fournisseur de services : plusieurs étapes intermédiaires de négociation sont prévues, assurées par le facilitateur, afin d'acheminer chaque requête à l'agent le mieux à même de la traiter. De même, de façon générale, les réponses ne sont pas adressées directement à l'auteur de la requête, mais amenées à la connaissance du facilitateur qui se charge de les acheminer au bon destinataire. Ce fonctionnement indirect est qualifié de « délégation » par les concepteurs de la plate-forme. Le principe de délégation implique que les agents, et par extension le développeur humain, n'ont pas à connaître précisément quel agent offre quel service, sur quelle machine. Le facilitateur prend en charge les interactions entre agents et la satisfaction des requêtes exprimées.

La délégation repose sur une spécification formelle, abstraite, tant des services pouvant être assurés par chaque membre de la communauté, que sur celle des requêtes. Les échanges sont ainsi régulés par le facilitateur par confrontation (unification) entre des requêtes et des services enregistrés auprès de celui-ci.

2.3 Déclaration d'un agent

2.3.1 Principaux types d'agents

OAA distingue les types d'agents suivants :

- Facilitator, centralise les requêtes et coordonne l'exécution des agents ;
- Interface, dédiées à la gestion des entrées multimodales (écriture manuscrite, voix, vidéo etc…) et à l'affichage ;
- Application ;
- Méta-agents, qui intègrent des connaissances afin de guider les autres agents (ex : algorithmes d'apprentissage, réseaux de neurones, systèmes-experts etc…) ;
- Langage naturel, à l'interface entre les requêtes en langage naturel et les requêtes ICL.

La figure ci-dessous donne une représentation schématique d'une communauté d'agents idéale.

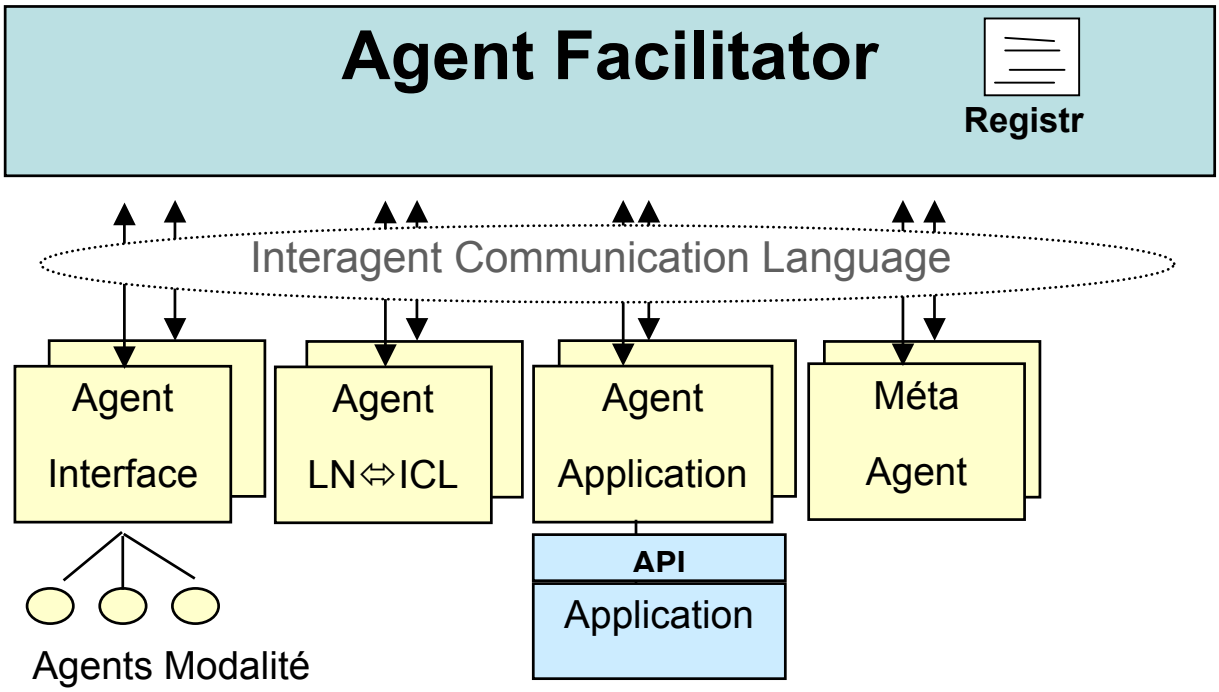

Figure 3 : communauté d'agents OAA idéale

2.3.2 Principales étapes

La création d'un agent OAA passe par les étapes suivantes, dont les détails sont fournis plus bas lorsque nécessaire :

1. déterminer quels services, ou *solvables* doivent être fournis par l'agent. Ceux-ci seront déclarés à la connexion avec le facilitateur et l'implémentation de l'agent sera structurée autour des *solvables* ;

2. inclure une copie de la bibliothèque Agent, en suivant les conventions propres à chaque langage. La librairie Agent est une API fournissant l'accès aux fonctionnalités de la plate-forme OAA, en vue du développement de systèmes multiagents ;

3. de façon optionnelle, surcharger (redéfinir) les comportements par défaut à l'aide d'appels spécifiques ;

4. la définition de *triggers* (voir plus bas) de type *data*, *task* ou *procedure* est effectuée grâce à `oaa_AddTrigger` ;

5. une procédure de *callback* doit être définie par chacun de ses *solvables procedure*. Le code, défini pour chacun de ces *callback*, est susceptible d'installer des *triggers* locaux ou distants, de lire ou d'écrire à partir du facilitateur, d'envoyer des événements ou des requêtes au facilitateur ou à un agent particulier, ou encore de s'interfacer avec des routines associées à l'agent ;

6. une communication avec le facilitateur doit être ouverte, par le biais de `com_Connect(parent, [], address)` ;

7. l'agent doit s'enregistrer auprès d'un facilitateur, *via* `oaa_Register(parent, AgentName , Solvable, Params)` ;

8. enfin, l'agent doit initialiser une boucle d'événement, *via* `oaa_MainLoop`.

2.3.3 Déclaration des services, ou « solvables »

Chaque agent membre d'une communauté OAA définit et rend publiques un ensemble de services qu'il est capable de rendre, exprimées en ICL : les *solvables*. Ces services mettent en place une interface de haut niveau entre l'agent et le facilitateur. Ce dernier peut ainsi déléguer les requêtes de service à l'agent.

Notons que les solvables sont typés :

- Procédures (procedure) ;
- Données (data) ;
- Triggers.

Les solvables *procedure* et *data* déclarent les services pouvant être appelés directement par d'autres agents (par l'appel de la procédure de la librairie Agent : `oaa_Solve`). Fondamentalement, un solvable *procedure* accomplit une action, alors qu'un solvable *data* enregistre un ensemble de faits. Ainsi, par exemple, un agent de courrier électronique définirait des solvables *procédure* afin d'envoyer un message au destinataire. De son côté un agent encapsulant une base de données définirait un solvable *données* correspondant à chaque

relation présente dans la base. Les solvables *data* sont le plus souvent utilisées pour fournir des enregistrement de données partagés, pouvant faire l'objet d'une requête mais pouvant également être mis à jour par les agents autorisés à le faire.

Les solvables *triggers* ne sont pas directement accessibles par les autres agents, qui ne peuvent y avoir accès qu'indirectement, en assignant des *triggers* de tâches aux agents concernés. Un solvable *trigger* n'existe que pour déclarer des conditions ou événements spécifiques à une tâche donnée. Pour reprendre l'exemple de l'agent de courrier électronique, celui-ci peut déclarer un solvable *trigger* surveillant l'arrivée de nouveaux messages, en fonction du profil des utilisateurs. Cette déclaration signifie aux autres agents (y compris le facilitateur et l'agent source) que des *triggers* peuvent être assignés à cet agent. Le type *task* est le seul trigger pour lequel une déclaration de solvable *trigger* est nécessaire, *via* l'appel aux procédures de la librairie Agent telles que : `oaa_Declare` et `oaa_Undeclare`.

2.3.4 Evénements

L'ensemble des communications entre agents ont lieu sous la forme d'événements. De plus, la plupart des traitements et des structures internes d'un agent sont normalement centrés autour de ces événements. On peut penser ces événements en termes de messages, qui ne doivent pas être directement construits par le développeur : la construction et la transmission de ces événements est le résultat des appels aux fonction OAA telles que `oaa_Solve` et `oaa_AddTrigger`.

Ainsi, un appel à

```
Oaa_Solve(Goal, Params)
```

au sein d'un agent A a pour résultat un événement de la forme

```
ev_solve(GoalID, Goal, Params)
```

de A vers le facilitateur, ainsi qu'un message de retour de la forme

```
ev_solved(GoalID, Requestees, Solvers, Goal, Params, Solutions).
```

2.3.5 Appels de services et traitement des requêtes

Un agent fait appel à des services par l'envoi de buts au facilitateur. Chaque but contient des appels à un ou plusieurs *solvable*. L'appel à un *solvable* n'implique pas la spécification d'un agent particulier pour le traitement de la requête. Bien qu'il soit possible de désigner un agent particulier pour le traitement d'une requête, il est généralement préférable de faire appel au facilitateur. Les fonctions de la bibliothèqe Agent d'OAA mettent un place un point d'entrée unique pour l'appel aux services d'autres agents : la procédure `oaa_Solve`. Cette procédure peut aussi bien servir à extraire des données qu'à déclencher des actions. Par ailleurs, elle fournit un certain nombre de paramètres différents permettant de contrôler le comportement du facilitateur et des autres agents. En particulier, le paramètre `address` permet la

délégation explicite envers un ou plusieurs agents. Ce paramètre permet également l'appel aux solvables de l'agent émetteur lui-même.

Les données qui sont retournées en réponse à une requête, ou *solvable data*, ainsi que celles utilisées pour les opérations de mise à jour sont directement exprimables en ICL. Ces données peuvent être complètement ou partiellement instantiées.

2.4 Exemples

Ci-dessous, deux exemples en Java et en C, adaptés du manuel de référence de la plate-forme OAA. Ces exemples illustrent les étapes données plus haut de création d'un agent.

2.4.1 Un agent fax en Java

```
import oaa.agents.bean.agentBean.*;

AgentBean oaa = new AgentBean();
```

1. Importation des librairies

```
oaa.setOaaName("fax");

oaa.setOaaSolvables("[fax(Destination,Document)]");
```

2. Déclaration des services

```
String oaa_doEvent(DoEvent e) {

   if (e.func.compareTo("fax") == 0) {

        String person = oaa.lib.nthElt(e.args, 1);  // Person

        String doc = oaa.lib.nthElt(e.args, 2); // Document

        String res = oaa.lib.solve("fax_num(" + person + ",N)",
"[]");

        if (res.compareTo("[]") != 0) { ...

      }

}
```

3. Définition des services

```
oaa.connect();
```

4. Initialisation

2.4.2 Un agent Text-To-Speech en C

```
#include <libcom_tcp.h>

#include <liboaa.h>
```

1. Importation des librairies

```
ICLTerm capabilities = icl_TermFromStr("[play(tts, Msg)]");
```

2. Déclaration des services

```
ICLTerm oaa_AppDoEvent(ICLTerm Event, ICLTerm Params) {
      if (strcmp(icl_Str(Event), "play") == 0) {
          return playTTS(icl_ArgumentAsStr(Event, 2));
      }
      else return NULL;
}
```

3. Définition des services

```
main() {
      com_Connect("parent", connectionInfo);
      oaa_Register("parent", "tts", capabilities);
      oaa_MainLoop(True);
}
```

4. Initialisation

3 Développer une application de TALN avec OAA : exemples détaillés

Dans cette partie, nous examinons la mise en œuvre d'applications dédiées au TALN. La première est une plate-forme de commande vocale d'interfaces (ex : simulateur tactique). Les applications présentées sont des systèmes industriels en vraie grandeur, dont nous détaillerons le cadre, les contraintes et les composantes retenus au cours de la présentation du tutoriel. Les applications présentées intègrent des composants de reconnaissance vocale, de cartographie, des grammaires destinées à analyser le langage naturel (grammaires génériques, spécifiques, règles d'interprétation contextuelle, résolution des anaphores etc…).

3.1 Thomspeaker, commande vocale d'interfaces

3.1.1 Architecture

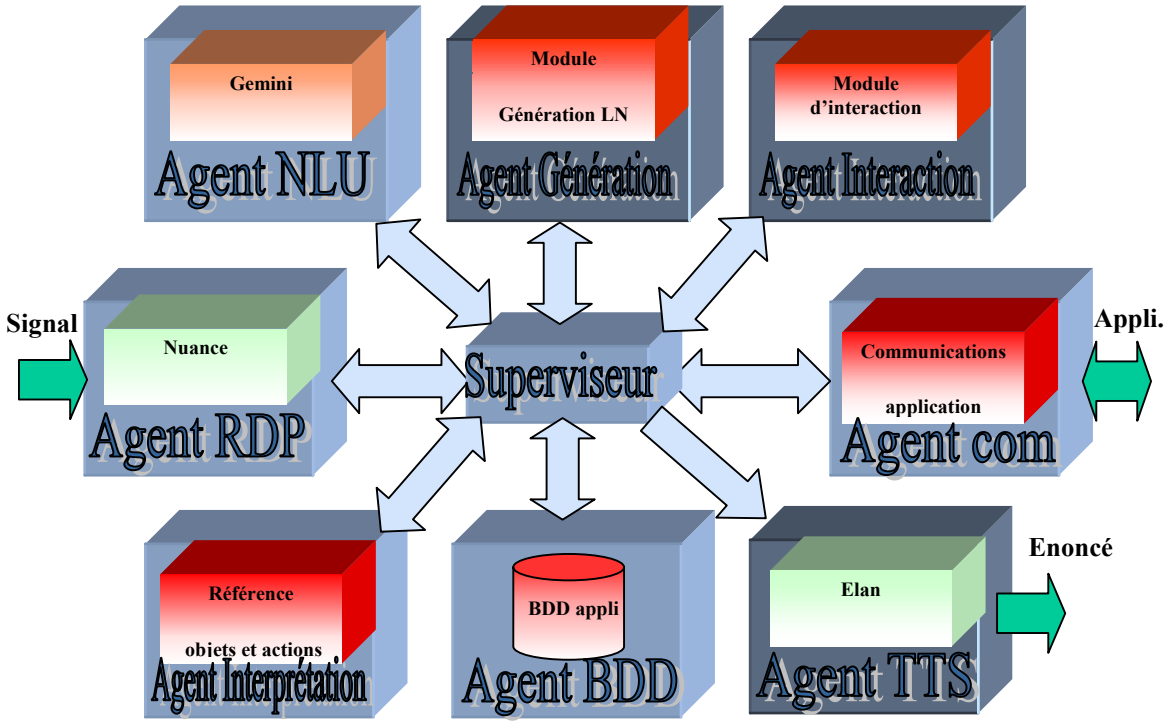

Figure 4 : architecture d'un système multiagents industriel, Thomspeaker

3.1.2 Fonctionnalités

- Grammaire générique de l'anglais pour la commande
- Grammaire générique du français pour la commande
- Grammaire de l'anglais pour les requêtes
- Module de négociation avec sous-dialogues
- Génération de réponses
- Synthèse de parole
- Interprétation contextuelle (théorie des RMs)

- Rattrapage des erreurs de reconnaissance au niveau interprétation
- Modification/restriction en dynamique de la phraséologie
- Multi-utilisateurs
- Outils de saisie des ressources

3.2 MARA, assistance à la maintenance

3.2.1 Architecture du système

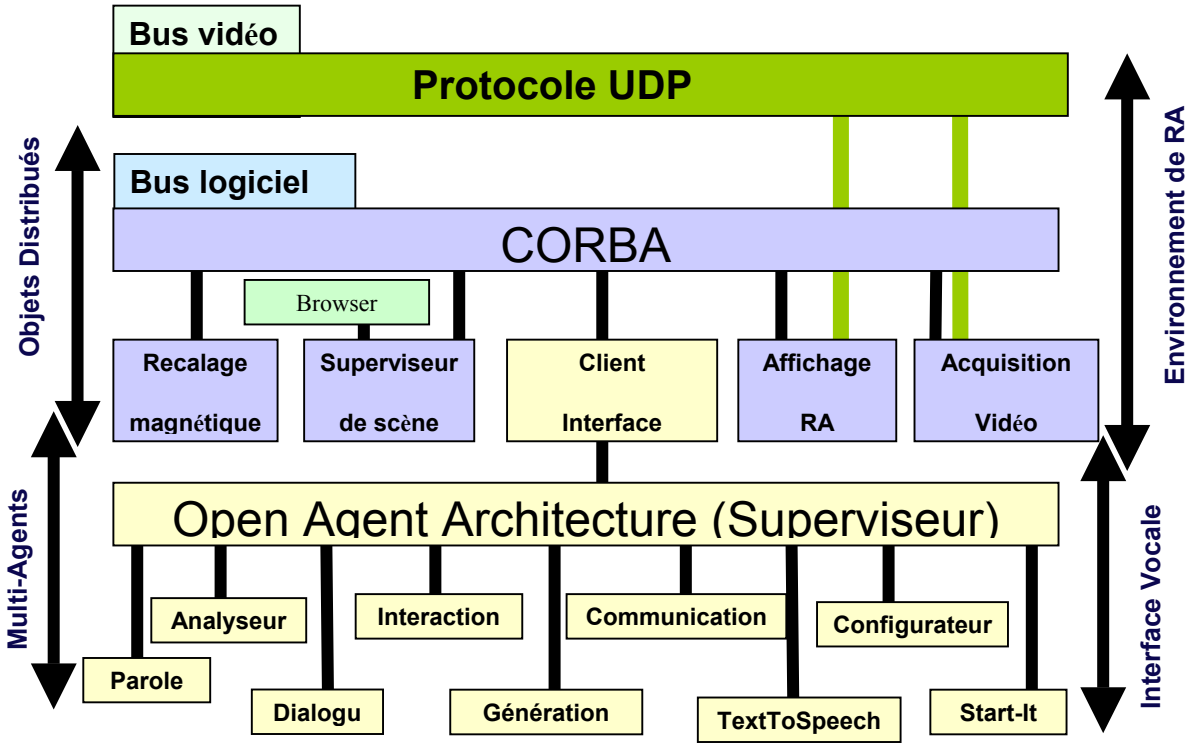

Figure 5 : architecture fonctionnelle d'un système d'aide à la maintenance

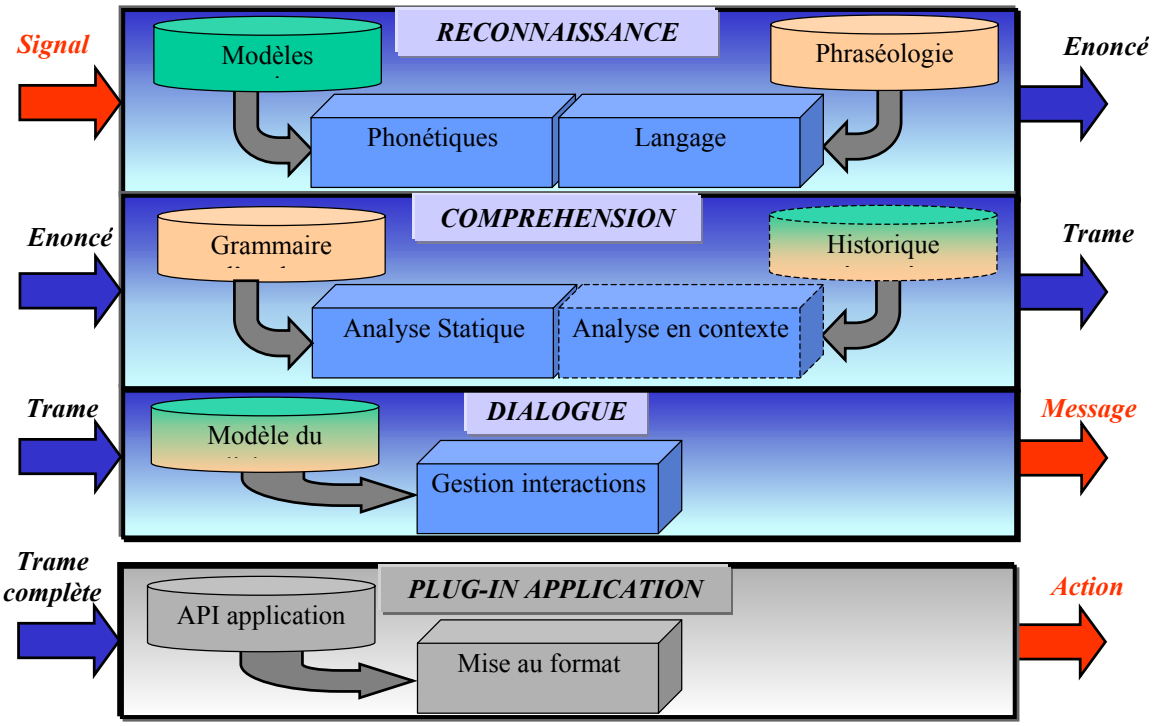

Figure 6 : sources de connaissances et traitements du système MARA

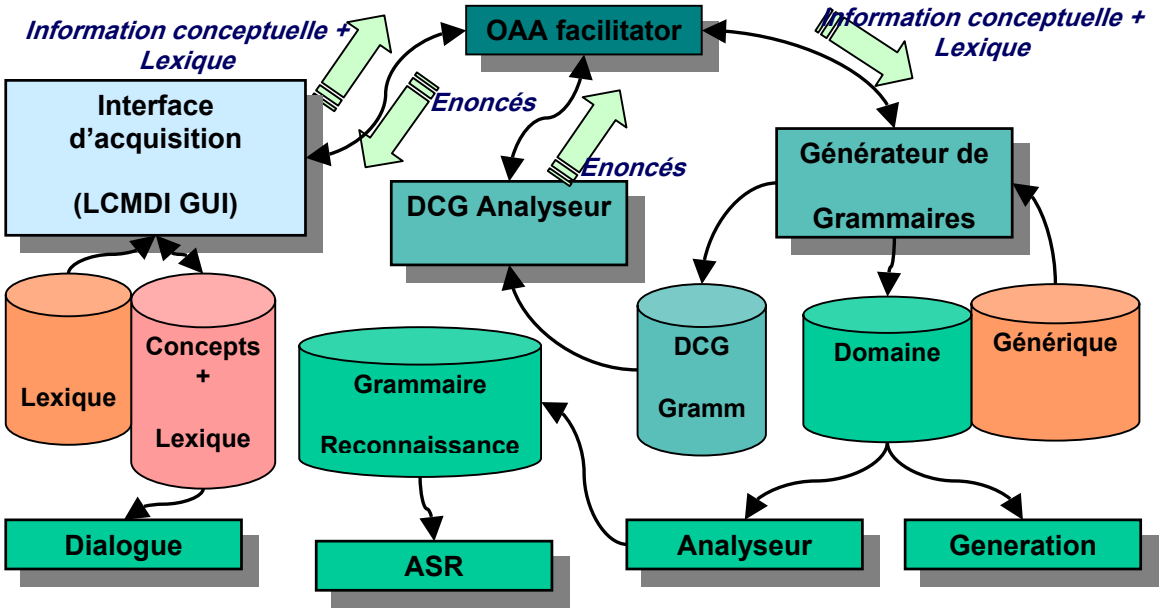

Figure 7 : architecture d'acquisition de ressources pour les systèmes industriels multiagents

Cette architecture est celle mise en œuvre pour un système destiné au développement de ressources linguistiques destinées à être utilisées par des systèmes multiagents utilisant la reconnaissance vocale, tels que présentés plus haut. Cette architecture est elle-même

multiagents et repose sur OAA. Les ressources sont générées à partir de ressources génériques, telles qu'une grammaire générique décrivant les principaux types de phrases, par exemple. La sortie du système est un ensemble de règles d'interprétation spécifiques à un domaine, produites à partir des ressources génériques et de ressources telles qu'un lexique-métier (ex : lexique de la maintenance), ou des modèles acoustiques en milieu bruité. Ces règles d'interprétation propres à un domaine de spécialité permettent de guider les modules de reconnaissance de la parole et d'interprétation, en éliminant les candidats improbables, étant donnée l'application.

3.3 Démonstrateur MARA

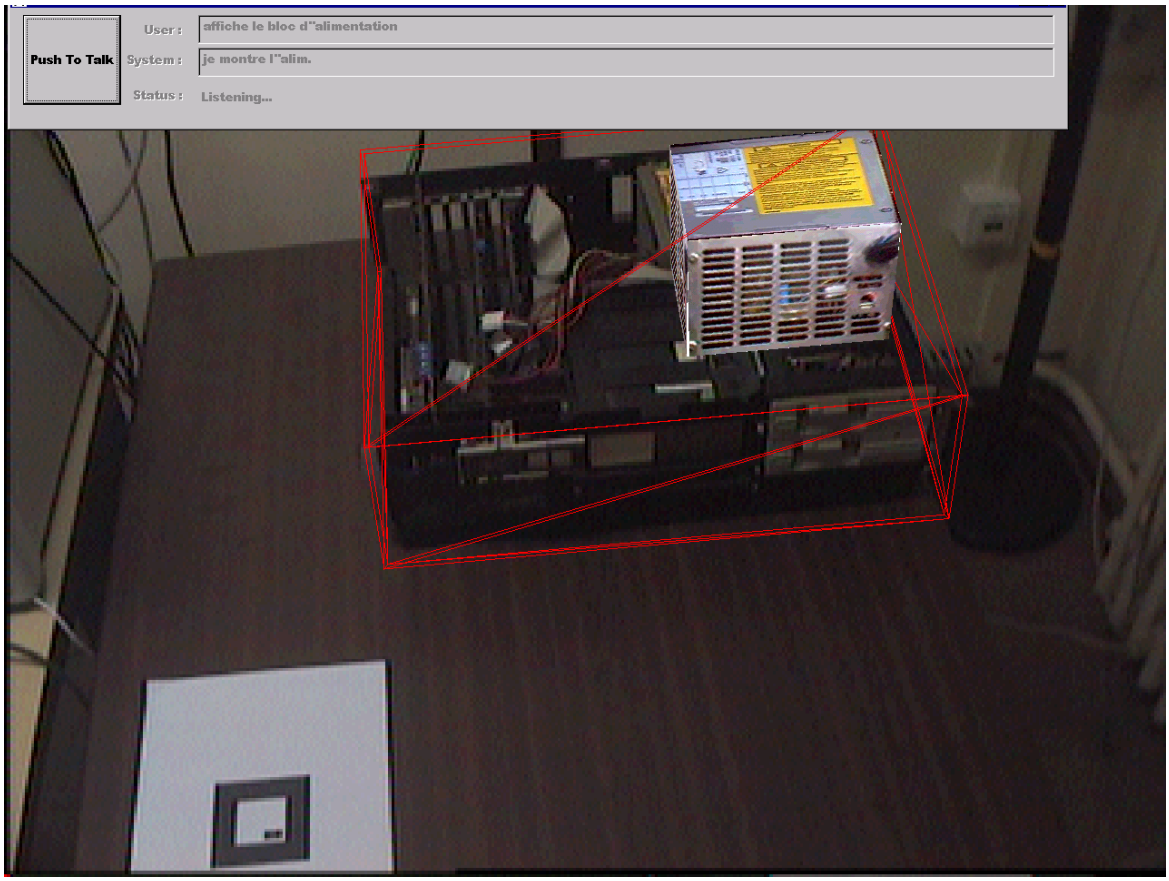

Figure 8 : interface de réalité augmentée du système MARA

La figure ci-dessus donne la vision augmentée fournie au technicien utilisant le système MARA de maintenance à distance, intégrant des fonctionnalités de reconnaissance vocale, de réalité augmentée (superposition d'informations graphiques), de gestion des documents (*i.e.* la documentation technique) : textes, vidéos, graphiques etc… Le système répond ici à une requête vocale de l'utilisateur : « afficher le bloc d'alimentation », en mettant en évidence le bloc d'alimentation.

L'ensemble des requêtes vocales sont traitées par un agent de reconnaissance vocale reposant sur Nuance (SRI), ainsi que sur une grammaire du domaine, exprimée de façon traditionnelle sous la forme de clauses Prolog (DCG).

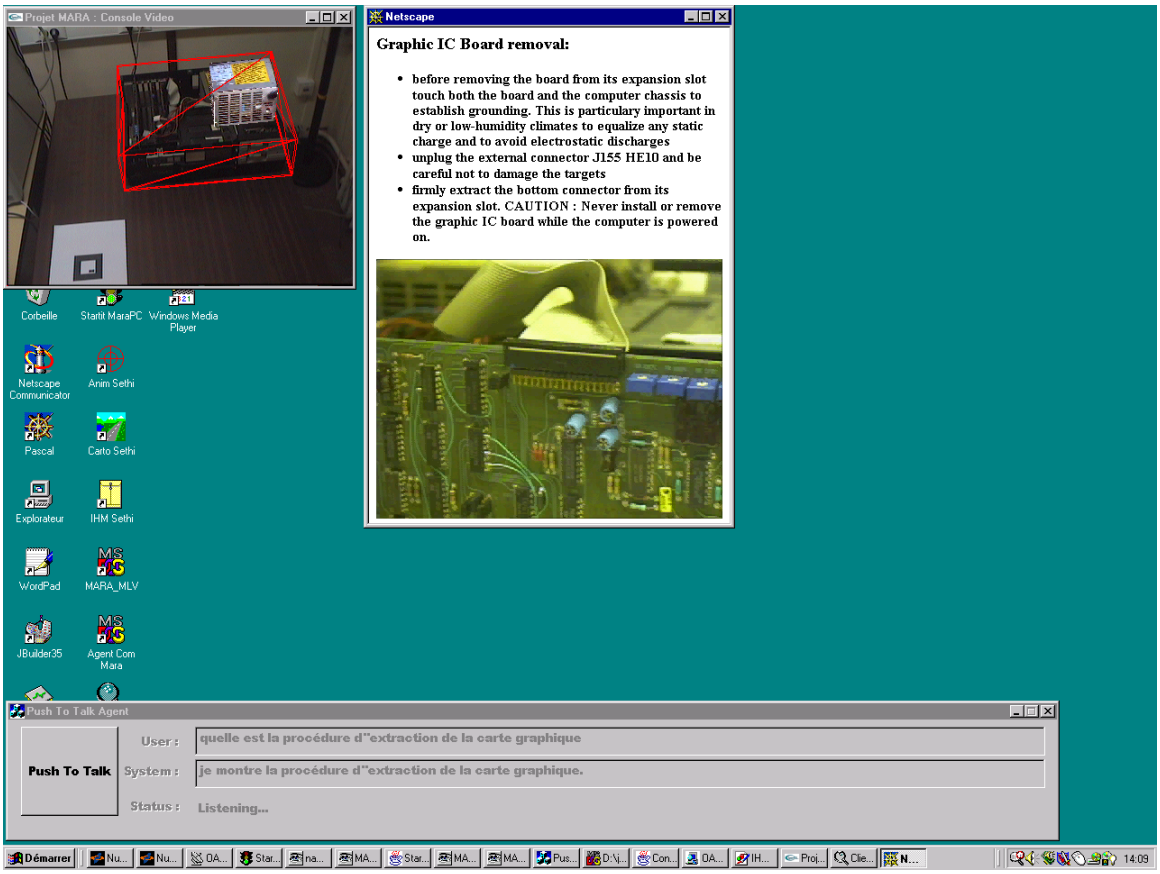

Figure 9 : interface homme-machine pour le système MARA, extraction de la carte graphique

La figure ci-dessus présente une autre phase de l'interaction : l'utilisateur demande « quelle est la procédure d'extraction de la carte graphique », qui suscite l'affichage d'un extrait de documentation technique multimédia illustrant la procédure demandée.

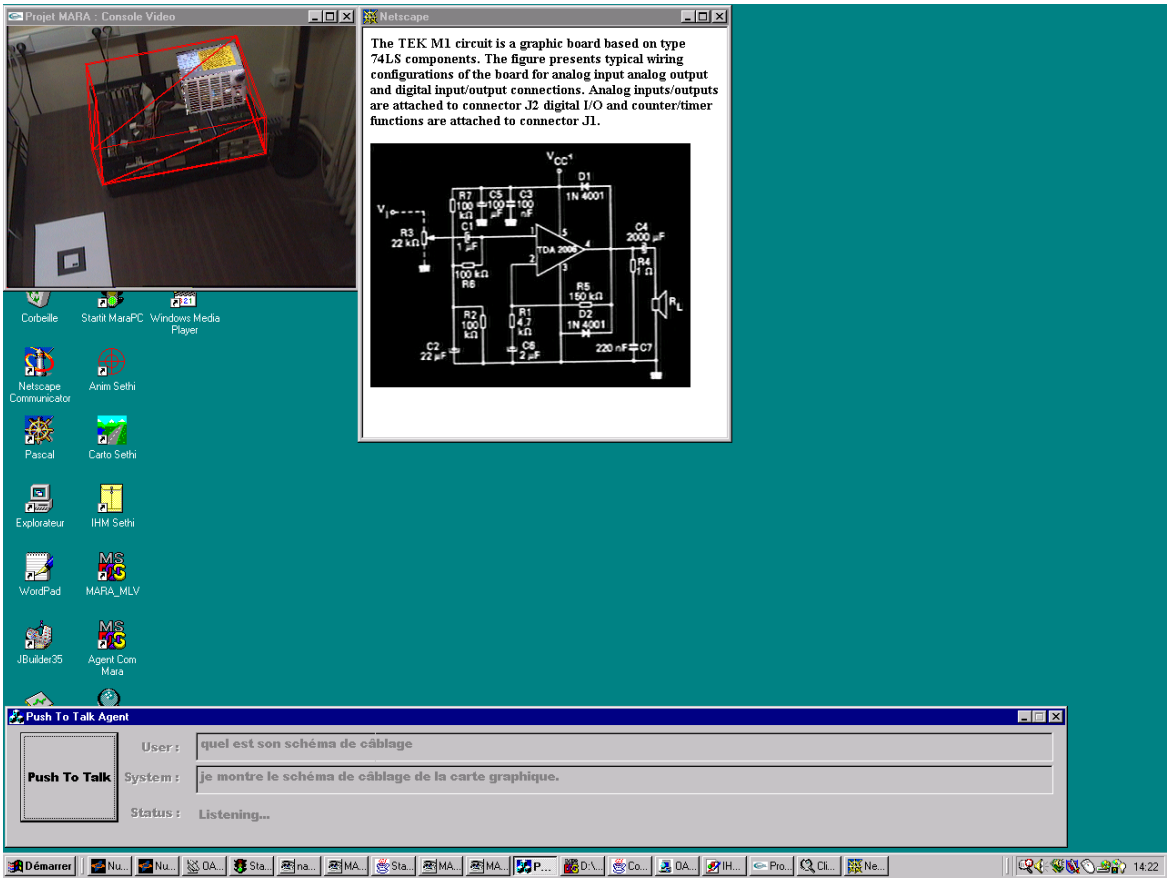

Figure 10 : interface du système MARA, résolution d'anaphores

La figure ci-dessus présente le schéma de câblage demandé par l'utilisateur *via* la question « quel est son schéma de câblage ». Le système fait appel à un agent de résolution des anaphores pour traiter la requête et affiche le schéma de la carte graphique.

Références

Cunningham H. *et alii.*, 1997. Software infrastructure for Natural Language Processing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, March 1997.

Cunningham H. *et alii.*, 2001. Developing language components with GATE, University of Sheffield.

Finin T., Labrou Y, Mayfield J., 1997, KQML as an agent communication language, in Jeff Bradshaw, editor, *Software Agents*, MIT Press, Cambridge.

FIPA, 1997, Foundation for intelligent physical agents, (FIPA) 1997 specification (http://drogo.cselt.stet.it/fipa/spec/fipa97.html).

Genesereth, M. R., Fikes R.E., 1992, Knowledge interchange format version 3.0, Reference Manual, Technical Report Logic-92-1, Stanford University (http://logic.stanford.edu/kif/kif.html).

Labrou Y., Finin T., 1997, A proposal for a new KQML specification, Technical Report CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, (http://www.cs.umbc.edu/kqml).

Martin D., Cheyer A.J., Moran D.B., (1999), The Open Agent Architecture: a framework for building distributed software systems, *Applied Artificial Intelligence*, vol. 13, p 91-128.

Martin D., Cheyer A.J., Moran D.B.. The Open Agent Architecture: a framework for building distributed software systems, Artificial Intelligence Center, SRI International.

Maynard D. *et alii.*, 2000. A survey of uses of GATE, Research Memo CS-00-06, Institute for Language Speech and Hearing (ILASH) and Department of Computer Science, University of Sheffield.

Open Agent Architecture (OAA) developer's guide, V2.0, SRI International.