Osvaldo Cairo L. Enrique Sucar
Francisco J. Cantu (Eds.)

MICAI 2000: Advances in Artificial Intelligence

Mexican International Conference
on Artificial Intelligence
Acapulco, Mexico, April 2000
Proceedings

Springer

Organization

MICAI 2000 was organized by the Mexican Society for Artificial Intelligence and the Acapulco Institute of Technology (ITA).

Conference Committee

Conference Chair:	Francisco J. Cantu (ITESM-Mty, Mexico)
	L. Enrique Sucar (ITESM-Mor, Mexico)
Program Chair:	Osvaldo Cairo (ITAM, Mexico)
	Luis A. Pineda (UNAM, Mexico)
Tutorials:	Marc Boumedine (ITESM-CCM, Mexico)
Workshops:	Humberto Sossa (IPN, Mexico)
Local Arrangements:	Blanca E. Lopez (ITA)
	Juan M. Rodriguez (ITA)
Publicity and Design:	Moraima Campbell (ITESM-Mty, Mexico)
Finance and Registration :	Leticia Rodriguez (ITESM-Mty, Mexico)

Advisory Committee

Felipe Bracho
Alan Bundy
Ofelia Cervantes
Anthony Cohn
Francisco Garijo
Randy Goebel
Adolfo Guzman
Robert de Hoog
Felipe Lara
Christian Lemaitre
Jay Liebowitz
Cristina Loyo
Donald Michie
Jose Negrete
Pablo Noriega
Judea Pearl
Antonio Sanchez
Xindong Wu
Wolfgang Wahlster
Carlos Zozaya

Program Committee

Leopoldo Altamirano
Matias Alvarado
Ronald Arkin
Gerardo Ayala
Richard Benjamin
Marc Boumedine
Carlos Coello
Helder Coelho
Rogelio Davila
Javier Diez
Robert de Hoog
Guillermo de Ita
Boris Escalante
Vladimir Estivill-Castro
Jesus Favela
Asuncion Gomez-Perez
Jose Luis Gordillo
Silvia Guardati
Adolfo Guzman Arenas
Leo Joskowicz
Natalia Juristo
Nicolas Kemper
Angel Kuri
Ana Maria Martinez

Manuel Martinez
Raul Monroy
Eduardo Morales
Guillermo Morales Luna
Santiago Negrete
Alberto Oliart Ros
Mauricio Osorio
Luis Pineda
Alexandro Provetti
Carlos Ramirez
Homero Rios
Jaime Siamo Sichman
Carlos Sierra
Rogelio Soto
L. Enrique Sucar
Manuel Valenzuela
Javier Vega
Marteen Von Someren
Toby Walsh
Alfredo Weitzenfeld Zdnek Zdrahal
Carlos Zozaya

Referees

Angelica Antonio
Josep-Lluis Arcos
Eva Armengol
Oscar Dieste
Pere Garcia
Mario Martin
Nelson Medinilla
Pedro Meseguer
Ana Maria Moreno
Andres Silva
Sira Vegas

Collocated Conferences

3º TAINA - Workshop on AI
TIARP - Iberoamerican Workshop on Pattern Recognition

Sponsoring Institutions

The American Association for Artificial Intelligence,
International Joint Conference on Artificial Intelligence,
The Mexican Society for Computer Science,
CONACYT REDII.

And support from

ITESM, ITAM, IPN-CIC, UDLA, CICESE, UNAM-IIMAS, LANIA, UV, BUAP
Corporate support from BARTEC, DigiPro, World Software Services, Softek, Microsoft, IBM.

We would like to thank Francisco Solsona, Julio Barreiro, Jose Galvez, and Erick Rodriguez for their excellent organizational support. Special thanks go to Patricia Mora and the Springer staff for editing this volume.

Multiagent Systems

Reasoning Under Uncertainty

Interaction of Purposeful Agents that Use Different Ontologies

Adolfo Guzmán, Jesús Olivares, Araceli Demetrio, and Carmen Domínguez

Centro de Investigación en Computación (CIC),
Instituto Politécnico Nacional (IPN). Mexico City
aguzman@cic.ipn.mx, cejaj@acm.org

Abstract. At CIC we have developed a model that enables multi-threaded agents that do not share the same ontology, to interact and interchange information among them.
The behavior of each agent is defined in a high-level language with the following features:

(1) Each agent and each interaction can be described by several sequences of instructions that can be executed concurrently. Some threads belong to an agent, others are inherited from the *scripts* which they play or perform.
(2) Of all the threads, the agent must select which ones to execute, perhaps choosing between contradictory or incompatible threads.
(3) The model allows communications between agents having different data dictionaries (ontologies), thus requiring conversion or matching among the primitives they use (§4).
(4) Some of the threads can be partially executed, thus giving rise to the idea of a "degree of satisfaction" (§6.2.1).
(5) The world on which the agents thrive suffers unexpected events (§3), to which some agents must react, throwing them out of their current behavior(s).

The model, language, executing environment and interpreter are described. Some simple examples are presented. The model will be validated using test cases based on real situations like electronic commerce, product delivery [including embedding agents in hardware], and automatic filling of databases (§6.2.2).

Keywords: *agent, ontology, unexpected events, multiple threads, incomplete execution.*

[ntroduction and Objectives

vorld has now many information systems and databases, each using different lictionaries and concept organizations.
n the field of agents, most agents are not built by *us,* but by *somebody else.* our agents will interact mostly with "unknown and never seen before" agents ng from all over the planet. These agents will no doubt have diverse goals, and xpress their wishes or purposes using different knowledge bases. Consequently, anisms and forms to exchange information and knowledge among heterogeneous ns are needed.

ro, L.E. Sucar, and F.J. Cantu (Eds.): MICAI 2000, LNAI 1793, pp. 557—573, 2000.
nger-Verlag Berlin Heidelberg 2000

For versatility, an agent may have several behaviors: algorithms or threads "how to be rich", "how to cross the street", "how to bargain", etc. An agent may executing simultaneously several of his behaviors –those that apply–. A probl arises: some statement in thread *a* may contradict or be inconsistent with anot statement in thread *b*. If so, which one is to be executed? For instance, I may executing both the thread *traveler* of the script *on vacation* and also the thre *customer* of script *at a restaurant.* Then, I discover that the clam soup will take t hours to be cooked. "Wait that long", may say the first script, while the other may s "hurry, you may lose the plane."

An agent must obey not only his own rules (behaviors, algorithms). He must a adopt the rules imposed to him by the scripts on which he takes part. For instance, he goes into a restaurant in the role *customer*, he must obey (most of) the rules t the script *at a restaurant* (in Marvin Minsky's sense) impose on the *customer*. He c not enter and start selling lottery tickets to other customers, although that may ma with his behavior for "how to become rich." In this way, an agent acquires *additio obligations* (additional scripts to execute) on top of those with which "it was born."

Also: some scripts are incompletely executed. I do not want to eat soup. If food becomes unpalatable, I may leave. Thus, agents may skip rules or pieces of co

Unexpected events will happen. I was ordering more bread, when an earthqu shook the place. What to do? In general, I can not have a program to handle ev possible exception (the "frame problem" of John McCarthy), but I must have m general rules. We plan to use the model of a Turing machine with two tapes, additional one contains "unexpected events" and has an infinite number of symb that are additional input to the (normal) behavior [Wegner, 96].

1.1 Related Previous Work; General

An agent is an asynchronous process or demon, part of a collection of sim processes, that have a common goal, and that are geographically as well temporarily dispersed [Guzmán 98]. [Maes 94] develops agent-based systems t inherit from the user authority to act in his behalf and to make certain decisions in autonomous and proactive (without explicit user intervention) manner. See a [Minsky 85].

In some cases, software agents are created with anthropomorphic characteristi such as emotions [Bates 94], personality [Moon 96; Lester 97; King 96] and creati [Boden 94]. Their purpose is to give friendly help to users and to better as applications such as information systems.

Applications of agents vary: we find prototypes for training [Grant 97], emerg design [Edmonds 94], manufacturing [Zita 97], teaching [Selker 94; Sanchez Ayala 98; Guzmán and Núñez 98], electronic commerce [Chavez 97; Noriega 9 military [Amori 92], information filters [Baclace 92], extending object based syste into agent based systems [Amandi 97; Conrad 97], agent programming with programming languages [Canfield 94], internet applications [Etzioni 94; Barret 9 virtual environments [Tu 94; Maes 95], among others.

1.1.1 Languages for Agent Execution

Declarative: KQML [Finin 93], is a message format and a message handling protocol to enable agents to interoperate and share knowledge in run time without any concern about the contents of the messages.

Imperative: TCL [Gray 97, Rus 97], is a high-level scripting language that enables a user to define mobile agents. The language is simple and imperative, and it is interpreted, in order to avoid malicious agents to cause danger in the host machine. It is embeddable in other applications and extendible with user-defined command TCL has some ways to sense hardware, software and other agents and then adapt itself, and change its behavior. Nevertheless, each define agent is not concurrent.

Tele-Script is also an important recent development in the area.

Java is multi-threaded, and is being considered as a target language of a compiler of LIA (Lenguaje de Interacción entre Agentes). We decided to create LIA, since it will allow us easier study of desired features. It differs from KQML in the sense that the former is not concerned with the content of the message and therefore makes no changes, nor equivalence among ontologies. From TCL, our proposal differs because we provide a language to emphasize the relationships among agents even in different ontologies. Both languages are not current.

1.1.2 Interaction Machines

In [Wegner 95], interaction machines are defined as extensions to the Turing Machine, to enable it to deal with incomplete specifications that can be completed interactively; it is concerned with the incompleteness of Godel's theorem. We plan to use this model, but we will organize the infinite collection of "unexpected" symbols into a finite tree of concepts, following the lines of CYC [Guha 94] and Clasitex [Guzmán 98].

1.2 Prior Research Accomplishments

ANASIN [Guzmán 94b] is a commercial product that gathers information in differed fashion, dispersed in a large region of time and space, in order to obtain strategic data, useful for decision making, from operational data sitting in files and tables located in computers installed at the work centers. The agents used in ANASIN share a single ontology or data dictionary. This described solution will be much better rendered and generalized in the work hereby proposed.

CLASITEX [Guzmán 98] is a program that finds the main topics in an article written in Spanish. It does not work on key words, but on concepts. It uses a concept tree. For this reason, it is capable of finding an article talking about shoes, even if the article does not contain such word, but it contains instead the words boot, moccasin, sandals, ..., even if these words may refer to other contexts or concepts: moccasin is also a tribe of American Indians, ...

CLASITEX++ [Beltrán *et al* 98] is a variant of Clasitex. It has a tree of concepts in English, and therefore it can analyze texts in Spanish and in English. It also has a larger tree and is written in C (Clasitex is written in UNIX shell, with sed, awk, yacc, and other utilities).

A new version of Clasitex (unnamed, by Alexander Gelbukh [Gelbukh 99, 99b], Head of the Natural Language and Text Processing Laboratory at CIC) offers several additional advantages over Clasitex. Mikhail Alexandrov, of same laboratory, is the author of a text-processing program akin to Clasitex, but with different theoretical underpinnings.

ACCESS TO UNFAMILIAR DATABASES [Guzmán 94] allowed a user familiar with the data base manager (Progress) but little familiar with the *ontology* or meaning of the key words, fields, values, file names, etc., to be able to access in a useful manner strange data bases. For instance, the user could request "give me all the graduate students studying Algebraic Topology in Germany." The system identifies the *atoms* of the user's model, and converts them (using the common sense tree described in CYC) to the corresponding atoms of the target data base, giving explanations to the user (in written Spanish) such as:

- I do not have graduate students in the database, but I have Master of Science students, Ph.D. students, and post-doctoral students.
- I do not have Algebraic Topology, but I have Mathematics.
- I do not have data about Germany, but I have about Heidelberg, Bonn, Köln, Berlin, ...

The program converts a query (posed in Spanish) into an equivalent query, in Spanish too, but where the atoms are over the ontology of the target data base.

The CYC Project [Lenat & Guha 89] tried to construct the common knowledge tree in order to solve big problems in Artificial Intelligence. A. G. worked in this project. CYC's contributions show that it is possible to form trees or taxonomies of specialized knowledge areas (which is what we intend to do here), in addition to classifying the common knowledge (goal that, due to its extension –between one and ten million concepts– was not achieved by the project). A.G. has built trees of specialized knowledge for the project "Access to unfamiliar data bases", for ANASIN (where the tree takes the form of a data dictionary) and for Clasitex.

DRAWING 3D ORGANIC MOLECULES [Olivares 95] takes as input statements of organic molecules that conform to rules of the International Union of Pure and Applied Chemistry, transforms them into a 3D graphic language, and shows them. It lets the user to apply graphical transformations (such as rotations) to enable him to analyze the structural aspects of the molecules. Given a molecule, it produces an equivalent expression, but "expressed" in another ontology.

[Olivares 91], accepts as input declarative statements that register knowledge into a semantic net. It uses imperative and interrogative statements to query the semantic net. The input statements are provided in a subset of natural language and transformed to enable the system to understand them.

The Injector of Agents [Martínez 98] places demons in remote places, using the available network. It works under a variety of network types, communication types, protocols, and it supposes an hostile or disordered environment in the host, therefore needing to send several agents to become familiar with the host's environment, so that the last agent reaching it is the agent one wishes to inject.

[Huhns 98] describes a scenario similar to the one we propose here, but with single-threaded behaviors. [Huhns 97] describes how a set of autonomous agents

cooperate to coherently management of information in environments where there are diverse information sources, that is carried out by means of a common ontology. See also other works by Huhns, Singh, Mahalingam in the references.

2 Model for Agent Interaction

Our world is closed, and agents interact only with other agents. An agent has a *purpose(s)* that he tries to reach by participating in interactions or scripts. To participate in a script, he also needs to have the necessary resources indicated by the script. He may partially reach his purposes; hence the *degree of satisfaction* (§6.2.1).

An external user defines the agents, their properties, the interactions and their properties, using the LIA language. During execution, instances of agents and interactions are created. An agent can change its own purposes.

2.1 Agents

Agents represent (or mimic) real world entities, and are born with some initial threads (behaviors), among them are a few threads to handle unexpected events. Agents may also "take" (execute) threads obtained from an script in which they participate; for instance, agent Juan Pérez may "take" the thread or role "customer" in the script "at a restaurant."

Agents are atomic and are composed of several execution threads, each corresponding to a behavior or role in a script. When an agent takes (plays, performs) a role from a script, such role should be free, that is, not assigned to another agent.

During execution, exceptions may arise (because some resource is depleted, or due to an unexpected event

Features and resources of an agent are diffident via internal variables (Figure 1).

Agent's name.
Internal variables.
Purposes (as given by internal variables).
Normal (born with) behaviors (threads). In LIA
Behaviors (threads) in LIA, acquired from scripts
Behaviors (born with and acquired) for unexpected events. In LIA

Fig. 1. Components of an agent

Purposes. An agent has one or more purposes, indicated by values of internal variables.

Threads in each agent. An agent is composed of several threads, each one specifies an action or behavior to follow. Not all threads need to be active. Example: thread "how to cross the street", thread "greeting a friend." Each thread is a LIA program.

Variables in each agent. Information representing the knowledge of an agent is shared among its threads. If a thread knows that "today is Thursday", there is no way to hide this information from its other threads, or that one of these know that "today is Friday."

Both agents and interactions possess properties that are established by giving values to global, regional, internal and local variables. States, resources, roles of agents, interactions and threads are defined with the help of these variables, as follows:

- Global variables define the state of the simulation environment, and are visible for all agents and interactions (and all their threads). Example: time of day.
- Regional variables complement the state of some agents and interactions and are visible for those that declare them. Example: where agents A and B shall meet.
- Internal variables define the state of each instance of an agent, and are visible to all its threads (and to nobody else). They are composed of a name, a value, and the time of availability. Some internal variables are used to indicate the *purposes* of the agent.
- Local variables define the state of a thread and are only available to that thread.

A time-stamp in global, regional and internal variables indicates their time of availability.

2.2 Interactions or Scripts or Theater Plays

[Riecken 94] describes agent and interaction models. In our group, research focuses on the interplay, and how two (or more) interacting agents may reach each one its purposes.

Scripts or Interactions contain roles. Each role is a thread. For instance, interaction "at a restaurant" has the following roles: customer, waiter, cashier, cook. Agents take available roles of interactions in order to try to reach their purposes. For instance, if my purpose is "to satisfy my hunger", I may take the role of "customer" at the interaction or script "at a restaurant." When an agent takes a role, the thread begins to be executed by the agent. A thread has access to all global variables, to the regional variables that the agent or the interaction possess, to internal variables of the agent, and to local variables of the thread.

As consequence of the participation of an agent in several simultaneous interactions, it may happen that at a given moment, two contradictory instructions should be executed. For instance, a thread may say "go to Chicago" and the other "go to New York." CONTRADICT, the Contradiction Finder (§6.2.1) will detect and handle the contradictions, causing for instance that a given thread be suspended, reprogrammed, modified or that other threads become active.

Unexpected events alter, too, the execution of a thread. An agent may sense some events (rain) but not others (descent in air pollution). They are handled by MEI (§3.1).

2.2.1 Threads in Each Interaction

Each interaction (Figure 2) consists of several roles that contain the requirements for their activation and the benefits that the agent obtains if he executes the role.

For a role to be assigned to an agent, the role should be free and the purpose of the agent must coincide with the benefits of the role. When purpose and benefits match, the prerequisites of the role are reviewed [observing the values of the global and internal variables of the agent] and, if the agent fulfils them, then the agent gets the role (and the role becomes "assigned") and begins executing it. Example: role "cashier" has pre-requisite "can count." Example of a prerequisite on a global variable: "time must be after 12 noon".

Roles may be similar; for instance, the script "at a restaurant" may have 17 roles of the type "customer" and two of the type "waiter."

Role One: customer	Role Two: waiter	Role Three: cook
Prerequisites or requirements	Prerequisites or requirements	Prerequisites or requirements
Benefits	Benefits	Benefits
Purposes of this thread or role.	Purposes of this thread or role.	Purposes of this thread or role.
Local variables.	Local variables.	Local variables.
Code in LIA. How to play this role.	Code in LIA. How to play this role.	Code in LIA. How to play this role.
Internal variables (relevant to the script or interaction)		

Fig. 2. Components of the interaction or script "At a restaurant"

Each instruction is scheduled for execution by inserting it in the queue with an stamped execution time. When this time arrives, all instructions with that time-stamp are executed.

2.2.2 Communication among Agents

Communication among agents has followed two approaches: (1) imperative languages, where the user specifies what the agent should do [Rus 97]; (2) declarative languages, used for information exchange, for instance, KQML [Finnin 93 and 94]. In LIA, communication is indicated by *accept* and *output* statements in each thread. For instance, role "customer" may *output* "I want some bread", which is *accepted* by an accept instruction in role "waiter". Inputs that an agent can sense are queue at its input port until consumption; unwanted inputs are consumed [accepted] but ignored

3 Handling Unexpected Events

Unexpected events are unforeseen happenings. Our model handles them through MEI, a machine of unexpected events sitting outside the agents' environment, that (1) produces the unexpected events at unexpected times, (2) find out which agents can *perceive* the unexpected events, (3) interrupts all the agent's threads, (4) decides which *reaction behavior* (if any) should be activated in response to the event, (5) reactivates some of the interrupted threads, (6) detects the end of the event, (7) may activate additional threads, and (8) usually stops the reaction behavior.

3.1 Parts of MEI (Máquina de Eventos Inesperados)

- Generator of Unexpected Events. It generates and sends unexpected events to the LIA environment, where agents interact. It generates at the beginning of time all the unexpected events (rain, earthquake, I met an old friend at a shop, ...), of the form {type of event, start time, end time, values related to the event (intensity of rain, say)}. It performs action (1) of above list.

UnexpectedEventHandler. It performs actions (2)-(8), when called by LIA's interpreter.

3. 2 Handling an Infinite Number of Unexpected Events

An agent has (by birth, by acquisition from a script in which he plays a role, or by learning) a small number of canned behaviors to react to unexpected events. He may know how to react to rain (reaction depends if he is in a hurry, if he carries an umbrella, and so on), but not how to react to a volcano eruption. On the other hand, there is an infinite number of *types* of unexpected events. Being impossible for an agent to be born with (or to acquire) an infinite number of reaction behaviors, agents share the *tree of unexpected events,* where each event has a LIA thread on how to react to such event (Figure 3). This tree is infinite, but –as we said– an agent (a) can *sense* or perceive only a subset of unexpected events, and (b) possesses only a small number of reaction behaviors. Thus, • an agent reacts only to events which he can sense, and • he uses the tree to find out which is the most specific reaction behavior (to the event) that he possesses, and uses it. For instance, if an agent does not have a "volcano eruption" reaction behavior, then he may probably use the "life threatening" reaction.

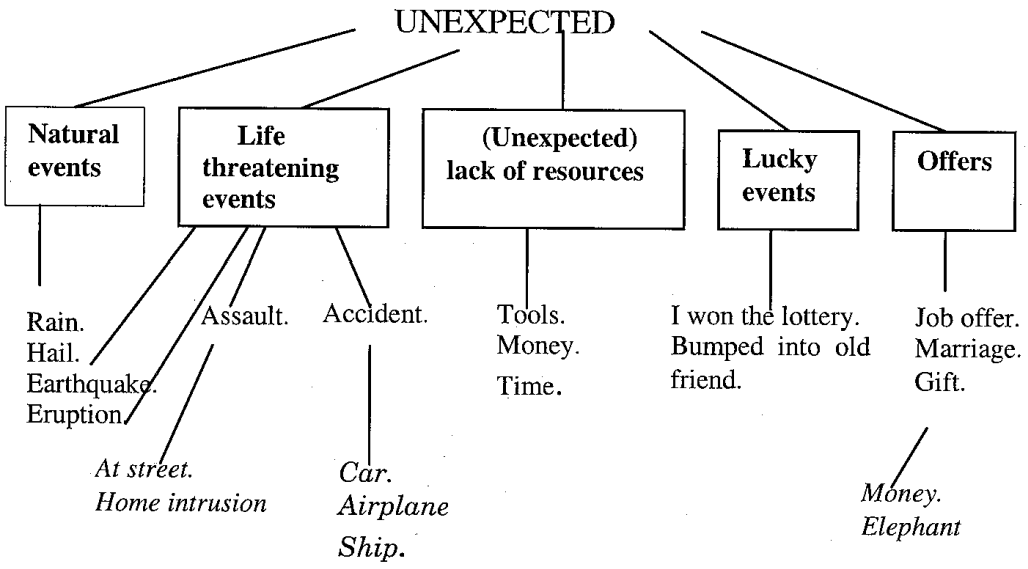

Fig. 3. Three of unexpected events.

The tree is really a lattice: a natural event (volcano eruption) can also be a life-threatening event.

Usually, a reaction behavior has parameters which define the precise reaction: Is the agent in a hurry? In a covered place? Does he carry an umbrella? Thus, depending on circumstances, an agent may decide to ignore *and not to react* to a sensed unexpected event: if he is being chased by a lion, he may ignore the rain.

4 Communication among Agents Possessing Different Ontologies

Agents written by all sorts of people must interact. For this interaction to be possible, two agents must share a common ontology (such as the common sense tree of CYC [Lenat & Guha 89]). Nevertheless, ontologies and depth of knowledge vary somewhat among agents. This section explains how to establish useful communication in spite of this.

For us, an ontology is a taxonomy (or tree) of *concepts* (not of words –thus, ontologies are language-independent). Since an agent can not transmit *concepts* to another agent,[1] he must use *words* in his preferred natural language. For our research, agents communicate through triplets of the form {entity; relationship; attributes} – notice similarity with E-R-A model of data bases– which roughly can be seen as {subject; verb; attributes}, for instance {Chevrolet car; sell; red, 1994, $5000}. To simplify our work, we assume that all agents share the same *verbs* (relationships, such as sell, buy, rent, and other verbs of electronic commerce), but the subjects may be different, such as car and automobile. We are also assuming that there are no mistakes in concept to word translation. Thus, "car" for an agent may not mean "airplane" for another agent, although "mole" for an agent may mean *the molecular weight of a substance, expressed in grams,* and for another agent it may mean *a spicy Mexican dish*, since the word mole has, in fact (as almost *any* word), several meanings or concepts: (1) a rodent; (2) a blemish of the skin; (3) a molecular weight; (4) spicy Mexican dish. As stated in [Guzmán 98], words are ambiguous; concepts are unique.

Each agent has a somewhat different ontology (a slightly different *dialect,* if they were languages). How is it possible for an agent to understand unknown words posed by the other agent? This is solved by COM, the Ontologies Matcher.

4.1 Matching Words Arising from Concepts in Different Ontologies

To explain how COM works, let us use the example that a push agent *A* issues the triplet {maize; sell; ton, $50} to a corresponding pull agent *B*. Two cases arise:

(1) *B* understands *maize,* that is, it has that word attached to one (or more) node in its ontology tree. *B* just has to ascertain the precise meaning that *A* wants to convey with *maize*. To do this, *B* goes to the father (generalisation) of *maize* in *B*'s ontology, and finds the concept *cereal,* which he transforms into the word cereal, which transmits (in the form of a question) to *A*: {questioning; cereal; meaning, father-of, maize}. *A* checks its own ontology, and finds that the father of *maize* is

[11] Once a global agreement regarding which concepts to share and how to structure the (shared, global) ontology is reached, concepts (nodes of the ontology) can be usefully transmitted.

also cereal. He answers {agreeing; cereal; meaning, father-of, maize}. Having two matches, both agents *A* and *B* are reasonably sure that they mean the same concept with *maize*. There is a match: *A* sells something that *B* most likely wants to buy. Perhaps they have to interact somewhat more, at the ontology level, to specify what color of maize *B* wants to buy, its size, etc. Perhaps *A* sells African maize, but *B* wants to buy Mexican maize.

If *A* were to issue {mole; sell; $5}, *B*'s question to *A* would be: {questioning; spicy Mexican dish, rodent, skin blemish, molecular weight of substance; father-of, mole}, to which *B* has to respond which of the four meanings is the intended one.

(2) *B* does not understand maize, so that he answers something like {questioning; maize; son-of, banana}, since *B* is an agent that buys grains and fruits, and he guesses that *A* wants to sell some kind of banana. *A* reacts as follows:

(2.a) If *A* has another word for the same concept, he issues it: {corn; sell; ton, $50}, which provokes a new call to COM from *B*.

(2.b) If *A* has no other word for the concept maize, he states that he wants to sell a particularisation of the father of maize (which is cereal): {sell; son-of, cereal; ton, $50}. To this, *B* must state that he is indeed interested in buying this kind of cereal (after all, *B* may be interested in buying heaters, not cereals). But he may or may not understand *cereal,* which may cause *A* to go to the father of cereal, *grain.*

(2.b.1) If *B* still does not understand grain, he probably is buying something else (heaters), and no communication is possible. *A* gives up.

(2.b.2) If *B* understands cereal, or grain, then basically *B* has to state that he is interested in buying this kind of grain or cereal: {buy, possibly; son-of, cereal; ton, $50}

To *B*'s answer, *A* must now *describe* this kind of cereal (or grain), namely maize, to *B*:

{sell; son-of, cereal, color, yellow, shape flat, size 1cm, ...; ton, $50}. This allows *B* to explore its own ontology, to see if *B* has some cereal with those features. Two cases:

(i) *B* has a concept matching that described by *A*, let us say, *corn.* Then *B* can decide whether he is interested in buying corn, and answers to *A* accordingly.

(ii) *B* does not have a concept matching *A*'s description. Two cases arise:

(1) *B* has a concept matching a subset of the values described by *A*. Then *B* proceeds as in case (i). For instance, *B* wants to buy a microwave oven, and *A* sells a brown microwave oven, 550 watts of power, model GE-530.

(2) *B* has no concept having a reasonable combination of features, of those explained by *A*. Then *B* gives up.

When *B* wants to match the properties of this cereal that *A* is selling, it may occur that the *description* used by *A* does not match exactly the description by *B*. For instance, *A* may use the word skin, as in color-of skin yellow, and *B* may use the word surface, or epidermis. Or, there may be a mismatch in some value: size 1 cm says *A*, while *B* understand inches; or round versus spherical . This provokes a recursive call to COM.

COM is needed because agents interact using natural words or particular concepts. The meaning of natural words is ambiguous. The meaning of a particular concept (such as good-quality-heater) is not universal. In both cases, the other agent needs to be sure which concept he is receiving. Once enough agreement is reached among grain merchants (people), as to exactly how to describe their grains, and what properties and values to use, grain agents can exchange concepts (on the *common* ontology tree agreed by the grain merchants), and ontology matchers like COM will no longer be needed. Also, the use of mark up languages, such as XML, can help in the matching of COM, since the agents can interchange the names (such as maize) and the ontology they use, this as a XML file.

5 The LIA Language

LIA (Lenguaje de Interacción entre Agentes) is used to define the simulation environment, the agents (§2.1) and the interactions (§2.2). Variables describe resources and purposes (Figure 1).
An agent is seeking to take part in interactions having roles that match his purposes. To take (play) a role, an agent must satisfy the requirements (prerequisites) established in the role.

5.1 The LIA World, Seen from Its Interpreter

During execution, instances of agents and interactions are created. Each time a role is taken, the *program counter* for that thread is initialized. As execution proceeds, each thread advances. A thread either runs to completion, or is stopped if some other thread has reached the purpose of *this* thread; or is suspended if there is resource depletion, unexpected events, conflicting threads, and in other cases. Threads can also be replanned (§6.2).

The LIA simulator or interpreter works in three stages:

A translator produces directives for the Constructor, out of LIA statements

The Constructor builds the data structures and organizes the roles so that agents can reference them. It handles main memory.

The Executor executes LIA statements, making use of MEI (§3.1), COM (§4), CONTRADICT (§6.2.1) and REPLAN (§6.2). During execution, events are displayed: creation of agents or scripts, reaching a purpose, unexpected events. Unknown LIA instructions are just displayed (printed). In this way, an instruction such as "I warmly greet agent b" gets displayed as "I, Juan Pérez, warmly greet agent b, Pedro Gómez", which helps in the understanding and debugging of the runs. Variables referenced in LIA commands must exist, otherwise an error message is issued, but execution continues. A table contains the duration of each type of instruction; otherwise, the interpreter uses a default.

At the end of a simulation, SATIS (§6.2.1) will tell the degree of satisfaction of each agent, and of each script. Simulation occurs inside a single process (and a single computer); that is, LIA agents are not traveling or spanning the network now.

5.2 Examples

At creation of an instance of an agent, the values of its internal variables and purposes are given, either in the user command line, or inside the LIA code.
Examples.

```
CREATE AGENT PERSON Juan_Perez, Money = 1000, Hunger = YES,
                    PURPOSES
                          Hunger == NO, To_become_rich == YES
CREATE AGENT PERSON Angel_Montiel,      Money = 0,
                                        Knows_serving = YES,
                    PURPOSES
                          Looks_for_work == YES
```

In capital letters appear LIA reserved words, other tokens are variables and values. For familiarity reasons, LIA's structure resembles C.

Example of an interaction (an auction). Buyers know in general the types of auctioned products, else they can ask for samples of the type of products, to decide whether to enter. The place is a physical place described by its properties. It is a closed place whose roof may collapse on earthquakes, but not subject to flooding. Agents know the regional variables pertaining to the auctioning place, once they enter. Buyers enter the auction because they seek some product; they can review the list of auctioned products. For a buyer to enter, it is required to have money or credit. Each buyer has a *prioritized* list of products, each with a minimum and a maximum purchase price.

For each actionable product, • the auctioneer announces the product and its initial price; • each interested buyer bids; • the auctioneer select the largest bid that surpasses the previous bid (of the last round). If after some time there are no more bids, the auctioneer makes the final countdown, and assigns the product to the best available bid. In case of earthquake, the auctioneer postpones the auction. He remembers the interrupted state of the auction, to resume it later if damages are inconsequential. A buyer detecting the earthquake postpones his purchase and executes his "earthquake present" reaction behavior (he may go out, go under a desk, start to cry, ...)

During auction, COM is used to render better understanding of the traded goods. An entity (name) at an input port of agent *A* triggers a call to COM, which resolves the meaning (finds the corresponding concept in *A*'s ontology) following the procedure of §0. If an agent needs to leave, he can not continue at the auction, unless the purpose satisfied by the auction has more priority than his purpose urging him to leave.

```
global
{
int importe = 123                    //Value of article
   ,contador                  //Counter
   ,importe ;
char articulo[10][20] ;
double precio[10] ;                  //Price
int maxprod = 10;                    //Max number of products
}
agente persona                 //Declaring an agent of type person
  regional                           //Regional variables
  {
      int temperatura ;              //Temperature
  }
  interna                            //Internal variables
  {
      char nombre[30] ;              //Name
      double dinero = 3000 ;  //Money
      int hambre = SI;        //Hunger = YES
      char lugar[40] ;               //Place
  }
  proposito                          //Purpose
  {
      hambre == NO;                  //Hunger == NO
      dinero > 3500;                 //Money > 3500
   }
   mei temblor()                     //MEI; earthquake
   {
       if( com(lugar ,"techado") == SI )          //If place is roofed
           lugar = "aire libre"; // go to an open (unroofed) place
   }
   mei lluvia()                      //MEI; rain
   {
        if( com(lugar ,"abierto") == SI )    //If place has no roof
            lugar = "techado"; // go to a roofed place
   }
   mei encuentradinero(int cantidad) //MEI; findsmoney (int amount)
   {
        dinero += cantidad;   //Add to my money the amount just found
    }
    mei pierdedinero(int cantidad)   //MEI; losesmoney (int amount)
    {
        dinero -= cantidad;   //Substract from my money the loss
    }
}
// DESCRIPCTION OF THE AUCTION
interaccion Subasta                  //INTERACTION; auction
{
   papel subastador(cupo 1,  //ROLE auctioneer, only one
       requisito tipopersona == "subastador",  //REQUIREMENT
       beneficio sueldo == 1000.00) //Benefits: high salary
    {
    int i ;

        // ANNOUNCES THE DEADLINE FOR REGISTERING
        finregistro = NO;            //End of registering = NO
        salida(finregistro ,RELOJ);  //Announces NOW that deadline for
                                     //registering is NO (not yet)
        tlimite = RELOJ + 01:00:00;  //Deadline is within one hour
        salida(tlimite ,RELOJ);      //Announces NOW the time of deadline
```

6.2.1 Theory

- Planning. As the result perhaps of unexpected events, current plans have to be modified or recalculated. This is handled by REPLAN.
- Detecting contradictory actions rests, among other things, on (a) detection of conflicts arising from use of common resources, such as time; (b) incompatible goals or purposes; (c) pursuing an unchanged purpose "for a while" (that is, not constantly changing the mind of an agent). J. O is constructing CONTRADICT, the Contradiction Finder, and REPLAN, the (re)planner.
- Measuring the degree of satisfaction of an agent, and of an interaction. It is performed by SATIS, a program that evaluates which purposes were reached, and to what degree.

Now, incomplete execution of a thread is possible: some threads may not run until completion, since they were stopped or suspended. Soon, some parts of a thread may be skipped (for instance, I do not want to eat soup, even if the role of "customer" in "at the restaurant" allows for soup eating. We plan to do this by marking part of the thread as optional. Also, time-outs may signal to skip secondary or less important parts of a thread.

6.2.2 Applications that We Want to Develop

- Electronic Commerce.
- Automatic programming via agents. An agent represents each code template (user interface template, data access template, ...). When presented a specification of a new program, agents look for work to do, and negotiate with each other their local interfaces.
- Automatic filling of databases. I just post what data I need; unknown agents supply me with it.
- Embedding agents into small cards. Smart articles. Distributed logistics. When my car passes the toll gate approaching Oaxaca City, its agent asks for directions to the Tecnológico de Oaxaca, to the agent in the toll gate.

Acknowledgements. Our gratitude to IPN and its director, Diódoro Guerra; to CONACYT and its director, Carlos Bazsdresch; and to REDII-CONACYT and its director, Dr. Felipe Bracho, sponsors of this project. Many ideas were suggested by or in conversations with Dr. Michael Huhns. Fruitful interaction occurred with members of the Natural Language Lab of CIC.

References

Those references marked with (•) are in Spanish.

Amandi, Analia and Price, Ana. (1997) Towards Object-Oriented Agent Programming: The Brainstorming Meta-Level Architecture. *Proc. of Autonomous Agents 97*, Marina del Rey, CA, USA

Amori, Richard D. (1992) *An Adversarial Plan Recognition System for Multi-agent Airbone Threats.* Computer Science Department, East Stroudsburg University.

Ayala, Gerardo and Yano, Yoneo. (1998) A Collaborative Learning Environment Based on Intelligent Agents. *Expert Systems with Applications* **14**, Number 1/2.

Baclace, Paul E. (1992) Competitive Agents for Information Filtering, *CACM,* No. 32.

Barret, Rob; Maglio, Paul P. and Kellem, Daniel C. (1997) WBI: A Confederation of Agents that Personalize the Web. *Proc. of Autonomous Agents 97,* Marina del Rey. CA

Bates, Joshep. (1994) The Role of Emotion in Believable Agents. *Comm. ACM*, **37**, No. 7.

Boden, Margaret A. (1994) Agents and Creativity. *Comm. ACM*, **37**, 7.

Canfield, Smith David et. al. (1994) KIDSIM: Programming Agents without a Programming Language. *Comm. ACM*, **37**, 7.

Chavez, Anthony and Maes, Pattie. (1997) *Kasbah: An Agent Marketplace for Buying and Selling Goods,* MIT Media Lab, Cambridge, MA.

Conrad, Stefan et al. (1997) Towards Agent-Oriented Specification of Information Systems. *Proc. of Autonomous Agents 97*, Marina del Rey, CA.

Etzioni, Oren and Weld Daniel. (1994) A Softbot-Based Interface to the Internet. *CACM*, **37**, 7.

Finnin, T.; Weber, J.; Widerhold, G., et al. (1993*) Specification of the KQML agent communication language* (draft). The DARPA Knowledge Sharing Initiative External Interfaces Working Group. http://www.cs.umbc.edu/kqml/kqmlspec/smecp.html.

Finin, Tim et. al. (1993b) *Specification of the KQML Agent Communication Language,* DARPA Knowledge Sharing Initiative, June 15.

Finin, Tim; Fritzon, Richard; McKay, Don and McEntire, Robin. (1994) KQML as an Agent Communication Language. *Proc. of the CIKM 94*, Gaitherburg MD, USA.

Finin, Tim et. al. (1994b) KQML as an Agent Communication Language. *CIKM 94* November, Gaitherburg, MD USA.

A. Gelbukh, G. Sidorov, and A. Guzmán. (1999) A method describing document contents through topic selection. *Workshop on String Processing and Information Retrieval,* Cancun, Mexico, September 22-24. 73-80.

A. Gelbukh, G. Sidorov, and A. Guzmán. (1999b) Document comparison with a weighted topic hierarchy. *DEXA-99, 10-th International Conference on Database and Expert System applications, Workshop on Document Analysis and Understanding for Document Databases,* Florence, Italy, August 30 to September 3. 566-570.

Grand, Stephen et. al. (1997) Creatures: Artificial Life Autonomous Software Agents for Home Entertainment. *Proc. of Autonomous Agents 97,* Marina del Rey, CA.

Gray, Robert S. (1997) Agent Tcl, in *Dr. Dobb's Journal,* March.

Guha, R.V. and Lenat, Douglas B. (1994) Enabling Agents to Work Together *CACM*, **37**, 7.

Guzmán, Adolfo. (1994) Project "Access to unfamiliar data bases". Final Report, IDASA. Mexico City. •

Guzmán, Adolfo. (1994b) Anasin. User's Manual. IDASA. Mexico City. •

Guzmán, Adolfo. (1998) Finding the main themes in a Spanish document. *Journal Expert Systems with Applications,* Vol. **14**, No. 1/2, Jan/Feb. 1998, 139-148. Handling of information in natural language (Clasitex).

Adolfo Guzmán and Gustavo Núñez. (1998) Virtual Learning Spaces in distance education; tools for the EVA Project. *Journal Expert Systems with Applications*, **15,** 34, 205-210.

Huhns, M. N. (1987) *Distributed Artificial Intelligence.* Pitman Publishing Ltd., London.

Huhns, M. N. and Bridgeland, D. M. (1991) Multiagent Truth Maintenance. *IEEE Trans. on Systems, Man, and Cybernetics,* **21**, 6, 1437-1445, December.

Huhns, M. N. and Singh, M. P. (1994) Automating Workflows for Service Order Processing: Integrating AI and Database Technologies, *IEEE Expert*, **9**, 5, 19-23, October.

Huhns, M. N.; Woelk, D. and Tomlinson, C. (1995) Uncovering the Next Generation of Active Objects, *Object Magazine*, **5**, 4, 32-40, July/August.

Huhns Michael N.; Singh, Munindar P. and Ksiezyk Tomasz. (1997) Global Information Management via Local Autonomous Agents, in *Readings in Agents,* M. N. Huhns, Munindar P. Singh, eds. Morgan Kauffmann Publishers, Inc.

Huhns, M. N. and Singh, M. P. (1997b) Internet-Based Agents: Applications and Infrastructure. *IEEE Internet Computing*, **1**, 4, 8-9, July-August.

Huhns, M. N. and Singh, M. P. (eds.) (1997c) *Readings in Agents*, Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Huhns, M. N. and Singh, M. P. (1998) Managing Heterogeneous Transaction Workflows with Cooperating Agents, in *Agent Technology: Foundations, Applications and Markets*, Nicholas R. Jennings and Michael J. Wooldridge, eds. Springer-Verlag, 219-240.

King, William Josseph and Ohya, Jun. (1996) The Representation of Agents: Anthropomorphism, Agency and Intelligence. *CHI '96* Companion, Vancouver, BC, Canada

Lenat, Douglas B. and Guha, R. V. (1989) *Building large knowledge-based systems.* Reading, MA: Addison Wesley. Ontologies for common knowledge. CYC project.

Lester, James C. et. al. (1997) The Persona Effect: Affective Impact of Animated Pedagogical Agents. *CHI 97*, Atlanta, GA, USA

Mahalingam, K. and Huhns, M. N. (1997) An Ontology Tool for Distributed Information Environments. *IEEE Computer*, **30**, 6, 80-83, June.

Maes, Pattie. (1994) Agents that Reduce Work and Information Overload. *CACM*, **37**, 7.

Maes, Pattie. (1995) Artificial Life Meets Entertainment: Lifelike Autonomous Agents. *Comm. of the ACM*, November 1995, **38**, 11.

Martínez-Luna, Gilberto. (1998) *Automatic installer of systems: mobile agents with blackboard structure.* Agent injection. M. Sc. Thesis, Departamento de Ingeniería Eléctrica (Computación), Centro de Investigación y Estudios Avanzados del I. P. N. •

Minsky, Marvin. (1985) *The Society of Mind.* Simon & Schuster Inc.

Moon Youngme, Nass Clifford. (1996) Adaptive Agents and Personality Change: Complementary versus Similarity as Forms of Adaptation. *CHI '96.* Companion, Vancouver, BC.

Noriega Blanco V., Pablo C. (1997) Agent Mediated Auctions: The Fishmarket Metaphor. Memory to obtain his Ph.D., Universidad Autònoma de Barcelona, Bellaterra. Spain.

Olivares, Jesús. (1991) *Evolutive System for Knowledge Representation,* B. Sc. Thesis at I. P. N.-Unidad Profesional Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas, Mexico City. •

Olivares, Jesús. (1995) *Drawing three-dimensional molecules.* EE Dept., Cinvestav-IPN. •

Riecken, Doug. (1994) M: An Architecture of Integrated Agents. *Comm. ACM*, **37**, 7.

Rus, Daniela; Gray, Robert and Kotz, David. (1997) Transportable Information Agents. *Proc. of Autonomous Agents 1997,* Marina del Rey, CA.

Sanchez J., Alfredo; Leggett, John J. and Schnase, John L. (1997) AGS: Introducing Agents as Services Provided by Digital Libraries. *DL 97.* Philadelphia PA, USA

Selker, Ted. (1994) COACH: A Teaching Agent that Learns. *Comm. ACM*, **37**, 7.

Singh, M. P.; Huhns, M. N. and Stephens, L. M. (1993) Declarative Representations of Multiagent Systems. *IEEE Trans. on Knowledge and D. E.*, **5**, 5, 721-739, October.

Tu, Xiaoyuan and Terzopoulos, Demetri. (1994) Artificial Fishes: Physics, Locomotion, Perception, Behavior.

Wegner, Peter. (1995) *Tutorial Notes: Models and Paradigms of Interaction,* Department of Computer Science, Brown University, USA, September.

Wegner, Peter. (1996) *The Paradigm Shift from Algorithms to Interaction,* Department of Computer Science, Brown University, USA, October 14th.

Wegner, Peter and Goldin, Dina (1998a) *Mathematical Models of Interactive Computing,* Draft on Observability and Empiricism.

Wegner, Peter. (1998b) *Towards Empirical Computer Science,* Brown University, USA.

Zita, Haigh Karen and Veloso, Manuela M. (1997) High-Level Planning and Low-Level Execution: Towards a Complete Robotic Agent. *Proc of Autonomous Agents 97.*