

An introduction to purposeful agents sharing 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

SUMMARY. A model and a language (LIA) are presented that allow multi-threaded agents to interact and interchange information, even if they do not share the same ontology. LIA is a high-level, multi-threaded language, that describes the behavior of each agent: (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) Some of the threads can be partially executed, thus giving rise to the idea of a “degree of satisfaction;” (3) Of all the threads, the agent must select which ones to execute, perhaps choosing between contradictory or incompatible threads; (4) The world of the agents is marred by unexpected events (§3), to which some agents must react, throwing them out of their current behavior(s); (5) The model allows communications between agents having different data dictionary (ontology), thus requiring conversion or matching among the primitives they use (§4).

The model, language, executing environment and interpreter are described. 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 that use different ontologies.

Key words: multiple threads, agent, ontology, unexpected events, incomplete execution.

1. INTRODUCTION AND OBJECTIVES

The world has now many information systems and databases, each using different ontologies, data dictionaries, and concept organizations. Agents must mimic, we believe, some of these features.

In the field of agents, most agents are not built by *us*, but by *somebody else*. Thus, our agents will interact mostly with “unknown and never seen before” agents arriving from all over the planet. These agents will no doubt have diverse goals, and will express their wishes or purposes using different ontologies. Consequently, mechanisms and forms to exchange information and knowledge among heterogeneous systems are needed.

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

An agent must obey not only his own rules (behaviors, algorithms). He must also adopt the rules imposed to him by the scripts on which he takes part. For instance, if he goes into a restaurant in the role *customer*, he must obey (most of) the rules that the script *at a restaurant* (in Marvin Minsky’s sense) impose on the *customer*. He can not enter and start selling lottery tickets to other customers, although that may match with his behavior for “how to become rich.” In this way, an agent acquires *additional obligations* (additional scripts to execute) on top of those with which “it was born.” Another point: some scripts are incompletely executed. I do not want to eat soup. If the food becomes unpalatable, I may leave. Thus, agents may skip rules or pieces of code.

Unexpected events will happen. I was ordering more bread, when an earthquake shook the place. What to do? In general, I can not have a program to handle every possible exception (the “frame problem” of John McCarthy): more general rules should exist. We use the model of a Turing machine with an additional tape containing “unexpected events” and has an infinite number of symbols that are additional input to the (normal) behavior [Wegner, 96].

1.1 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 located in the work centers. The agents used in ANASIN share a

single ontology or data dictionary, and this part of the 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,

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 94] 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” [Guzmán 94], for ANASIN (where the tree takes the form of a data dictionary) and for Clasitex [Guzmán 98].

[Huhns 98] has a scenario similar to the proposed here, but with single-threaded behaviors. [Huhns 97] describes how a set of autonomous agents cooperate to coherently manage information in environments where there are diverse information sources, that is carried out using a common ontology.

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.

2. HOW AGENTS INTERACT: A MODEL

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*.

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.

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 to a 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 defined via internal variables (Figure 1).

Agent’s name.
Internal variables.
Purposes (as given by internal variables).
Normal (born with) behaviors (threads). In LIA

Figure 1. Components of an agent

Behaviors (threads) in LIA, acquired from scripts
Behaviors (born with and acquired) for unexpected events. In LIA

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

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.”

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.

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.

2.1 Interactions or theater plays or scripts

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 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).

2.1.1 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)		

Figure 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.

3. UNEXPECTED EVENTS

Unforeseen happenings are handled through MEI (máquina de eventos inesperados), 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, the Unexpected Events Machine

- 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. 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.

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 94]). 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.

¹ 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.

5. THE LIA LANGUAGE

LIA (Lenguaje de Interacción entre Agentes) is used to define the simulation environment, the agents and the interactions (§2.1). 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.

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. CONCLUSIONS

Work has just begun. The interpreter has been completed recently by J. O. MEI is being programmed by C. D., and COM by A. D. So far, it seems that we made a good choice in having a multi-threaded language which we can freely change. Later, we may write a compiler from LIA to JAVA. Also, the assumptions under which this work is being carried look reasonable and interesting: • agents interact in stereotyped manners called scripts; • agents communicate with mixed ontologies; • unexpected events alter the normal interactions. Work continues. More features will be added to LIA. Applications will begin to test the language.

6.1 References. Those references marked with (•) are in Spanish.

- 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.
- 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.
- Guzmán, A., Olivares, J., Demetrio, A., and Domínguez, C. (2000) Interaction of purposeful agents that use different ontologies. In *Lecture notes in Artificial Intelligence # 1793, MICAI-2000: Advances in Artificial Intelligence*. O. Cairo et al. (eds). Springer Verlag. 557-573.
- 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., 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.
- Lenat, Douglas B. and Guha, R. V. (1989) *Building large knowledge-based systems*. Reading, MA: Addison Wesley. Ontologies for common knowledge. CYC project.
- Martínez-Luna, Gilberto. (1998) *Automatic installer of systems: mobile agents with blackboard structure*. Agent injection. M. Sc. Thesis, E. E. Dept., (Computer Science), CINVESTAV-IPN. Mexico City. •
- Minsky, Marvin. (1985) *The Society of Mind*. Simon & Schuster Inc.
- Olivares, Jesús. (1996) *Construction of a pictographic data base with application to collection of microorganisms CDDB-500*. M. Sc. Thesis, EE Dept., Cinvestav-IPN. •
- Wegner, Peter. (1996) *The Paradigm Shift from Algorithms to Interaction*, Department of Computer Science, Brown University, USA, October 14th.