# Agents Communicating for Dynamic Service Generation

Clement Jonquet, Stefano A. Cerri

# Agents Communicating for Dynamic Service Generation

**Clement Jonquet and Stefano A. Cerri**

LIRMM, CNRS & University Montpellier II
161 Rue Ada, 34392 Montpellier Cedex 5, France
{cerri,jonquet}@lirmm.fr

**Abstract**

This article proposes both an agent representation and an agent communication model based on a social approach. By modelling Grid services with agents we are confident to be able to realise the interactive, dynamic generation of services that is necessary in order to have learning effects on interlocutors. The approach consists of integrating features from agent communication, language interpretation and e-learning/human-learning into a unique, original and simple view that privileges interactions, yet including control. The model is based on STROBE and proposes to enrich the languages of agents (Environment + Interpreter) by allowing agents to dynamically modify them – at run time – not only at the Data or Control level, but also at the Interpreter level (meta-level). The model is inscribed within a global approach, defending a shift from the classical algorithmic (control based) view to problem solving in computing to an interaction-based view of Social Informatics, where artificial as well as human agents operate by communicating as well as by computing. The paper shows how the model may not only account for the classical communication agent approaches, but also represent a fundamental advance in modelling societies of agents in particular in dynamic service generation scenarios such as those necessary today on the Web and proposed tomorrow on the Grid. Preliminary concrete experimentations illustrate the potential of the model; they are significant examples for a very wide class of computational and learning situations.

**Keywords:** Interaction, Agent Communication, Grid Services, Service Generation, Partner Model, e-learning, Multi-Agent Systems, Grid, STROBE model, Social Informatics

# 1 Introduction

## 1.1 Learning, interaction and language

Assuming that the purpose of e-learning is to create/generate the conditions enabling and facilitating to improve human knowledge, we do not propose in this paper an Intelligent Tutoring Systems (ITS) able to directly do this job. Rather, we propose to reflect on interactions between entities, humans or computers, considering that these interactions are the ”the key of arch” of the knowledge construction occurring in e-learning scenarios. When humans use a computer they interact with a system. But could we really speak of ”interaction”? An interaction between two entities is a process that implies an action to occur on the interacting entities. That means that interactions have some consequence on these entities, i.e. a change of state. For humans, this change can be ”learning”, the definite purpose of ITS. In e-learning scenarios, it is quite unlikely that human learning occurs on the simple basis of interacting with a static system. Real interactions modify both entities, including the artificial one. We believe that the learning process is a co-construction of knowledge (social constructivism). If we want this process to be cumulative/unlimited in knowledge production, the two entities have to learn from each other during this process and re-inject what they learned into the loop. Consider, for example, an interactive CD-Rom on a special domain. The knowledge that the user of the CD-Rom would get is inevitably finite and limited due to the fact that the CD-Rom is a fixed support unable to change and to evolve to acquire new knowledge to offer to the user. Today this support is more or less obsolete; however its principles are still quite diffused, and e-learning supports are most often content centred: little effort is reserved in e-learning to the evolution of knowledge, in the system, as a consequence of interactions with the users. Dynamic, evolving systems apt to improve their capacity

to stimulate human learning, should be modelled accordingly. Interactions of these systems with humans – as well as with other systems in the network – should produce improvements of their ability to serve humans to learn. Probably the most advanced experiences on interactive, adaptive and evolutionary software entities come from the Distributed Artificial Intelligence (DAI) community where these entities are called agents. This paper deals with agents. In the rest of the paper, for the sake of simplicity, we will consider two types of agents: humans as Human Agents (HA) and computers as Artificial Agents (AA).

Therefore, the interest of this work for the e-learning/ITS community is justified by the assumption that modelling and making possible AA-learning helps to improve HA-learning (e-learning). Learning, for any kind of agents, is dependent from interactions.

The bi-directional interactions we are interested in are direct messages sent from an agent to another one and are called communications. In order to communicate, agents need a communication language understood by all the interlocutors. Therefore the problem of the common, shared, language appears. Most efforts both within the agent community and in research on Web or Grid Services assume a standard communication language to be adopted. We rather weaken this constraint, assume just a kernel common communication language to be adopted and propose to enrich the communication language at run time (dynamically). We propose a model which helps agents to build their own languages while communicating.

A language is basically a pair consisting of: i) a language expression evaluation mechanism and ii) a memory to store this mechanism and abstractions construct with the language. We call them the interpreter and the environment (as in the classic tradition of programming languages). In our model, agents are able to interpret messages in a given environment[1], with an interpreter both dedicated to the current conversation[2]. We will show how communication enables to dynamically change "values" in an environment, and particularly how these interpreters can dynamically adapt/change their way of interpreting messages. We call these evolutionary structures Cognitive Environments and Cognitive Interpreters. **Our agents therefore develop, while communicating, a language (Cognitive Environment + Cognitive Interpreter) for each of their interlocutors**.

This paper deals with learning for communicating agents. In a distributed system such as the Web and the Grid, three types of interaction between entities exist, and all of them may imply learning: i) HA-HA, via special devices such as for example enhanced presence as presented in [EKD03]. ii) HA-AA – the classical human-machine interfaces – when an agent requests another one for something, a service or a product (performing a task, answering a question, solving a problem). iii) AA-AA, when autonomous and intelligent AA collaborate with each others (e.g. by delegating a sub-task to another). In this paper our focus is on AA-AA as well as HA-AA interactions in dynamic service generation, but the model is generic with respect to any agent to agent interaction.

## 1.2 Grid services as agents

Current advances in agents and "agenthood" are reported in the MAS (Multi-Agent Systems) literature, for example [Fer99]. However, most of MAS literature, is concerned mainly with MAS consisting of AA. Our ambition goes beyond, includes HA within what we may call MAHAS (Multi Artificial and Human Agents Systems). A MAHAS is a system where AA and HA could interact and exchange information and knowledge easily, with no constraining limitation related to the nature of an agent. Questions (problems to solve) and their solutions emerge from this kind of system through interactions. Some should say that Internet could be considered as the first rudimentary, yet quite impressive, MAHAS in history. It is only partially true due to the product delivery aspect of the Internet network: HA and AA on the Internet deliver each other "ready made informational products", more than real services. Basically, Internet allows for the moment to store and retrieve "products" (e.g. pages, software, images) but not yet to participate to an agent dynamic society. Actually, surfing on the Web is equivalent to ask to another computer (on the network) to execute a calculus specified by an algorithm, or to give an answer to a question. The classical and most widespread Web architecture, the Client/Server one, represents this view. **We think that in order to become a real MAHAS, the Web should provide services instead of products, what is the aim of the Grid.** We further think that the Grid [FK99, FKT01, RJS01] can be seen as the evolution of both Web and agent research. For example, you cannot pass from a Client/Server model based network (as the Web) to a distributed resource sharing system (as the Grid) without considering autonomous and intelligent entities on this network. Even the simplest functionality claimed by the Grid – the availability of information resources, including computer power, in a way transparent to the Grid user – needs the user to fully delegate to the Grid the generation

[1]The term environment is here used with its programming language meaning, that is to say, a structure that links variables and values. It does not means the world surrounding an agent as in MAS community. This meaning will be used by default in the rest of the paper.

[2]A conversation is a set of communications, each consisting of at least two messages.

of the service, by accepting at the same time that the Grid adapts dynamically to the run time conditions in order to offer the best service to the user. This delegation is absent in classical Client/Server architectures.

The essence of the Grid concept is nicely reflected by its original metaphor: the delegation to the electricity network to offer me the service of providing me enough electric power as I need it when I need it even if I do not know where and how that power is generated. In [FKT01], Foster used the metaphor in the context of computing power: if I have a resource demanding application, I connect to the Grid; it will be the last that dynamically distributes the workload among the available computing power suppliers, returning to me just the outcome of the heavy computation and the bill for the consumed computational resources. This view considers:

**a.** Just one service, the computing power for heavy computations;
**b.** The concept of full delegation of the identified service to the Grid;
**c.** The concept of dynamically allocating and reallocating processes among available computing resources;
**d.** The concept of billing.

The subsequent refinement of Grid [RJS01, FKNT02] extends **a.** to any services. Particularly to services requiring to identify the semantics of message exchanges. Further, notice that in the initial concept of Grid, the interactive behaviour of Grid software concerns mainly the service delivery phase (the dynamic allocation of Grid resources is realised by a continuous interactive monitoring of the workload on the Grid). The service definition phase being quite simple (the request for power is clear and known straight ahead by the customer and by the Grid).

However, the notion of Grid Services such as described in [FKNT02] do not fully correspond to the service requirements of the Grid. Grid Services are quite the same as Web Services[3], but adapted to the Grid. Web Services are based on the four XML-like standard languages: i) WSDL (Web Services Description Language) to describe software components, ii) SOAP (Simple Object Access Protocol) to describe methods for accessing these components, iii) UDDI (Universal Description, Discovery and Integration) to identify the service provider in a service registry, iv) WSFL (Web Services Flow Language) to describe Web Services compositions. Besides, due to the emergence of Grid architectures, an evolution of Web Services to Grid has been proposed with OGSA (Open Grid Services Architecture) [FKNT02] and more recently WSRF (Web Services-Resource Framework). Web Services denote both an already used and a promising approach under one simple important viewpoint: it has facilitated the integration of various heterogeneous components. However, the concept of ”service” as outlined above is hardly to be recognised. The practice of current Web Services is still currently influenced by RPC-like (Remote Procedure Call) and CORBA[4] (Common Object Request Broker Architecture) approaches. Therefore, it has some well recognised weaknesses, as dynamic interoperability, composition, user adaptation, memory. For example, it does not take into account the autonomy of components, or it does not use a high level interaction language. Web Services do not care about persistency of the service. The notion of conversation (and its history) is absent in the Web Services approaches. For all these reasons, we think that Web Services is the first step of a service based network, but it can not be the last.

Foster et al. call service: *a (potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization* [FKNT02]. We actually believe such a service to be better called an ”agent”. The notion of service on the Web has to surpass HTTP protocols and XML standards being enriched by DAI research and especially by agent communication progress/improvement simply because the agent paradigm differs from the object one for three important ambitions of agents: autonomy, intelligence and interactive behaviour. Agents on the Grid have not simply to provide services; they must allow dynamic service generation.

## 1.3 Dynamic service generation

We mentioned above that today the Web may be considered an excellent mean for delivering informational products, but a quite limited tool for service generation. One of the fundamental differences between a product and a service is that when a client asks for a product, (s)he exactly knows what (s)he wants. A typical procedure call (with good parameters) is then realised just like in the RPC paradigm. In other words, the client asks a fixed algorithm (procedure) to be executed. At the opposite, when a client requires a service[5], (s)he does not exactly know

[3] http://www.w3.org/2002/ws/

[4] CORBA is a very general and open industry standard for working with distributed objects, it is to object oriented computing what the Web is to documents.

[5] This is a language abuse because we can not say that we require a service. The service is the result of the service generation process. We would have say ”when an user ask a service to be generated” or ”when a service is generated for a user”.

what (s)he wants and his/her needs and solutions appear progressively with the dialogue with the service provider until (s)he obtains satisfaction. The service generation approach does not assume the user know exactly what the service provider can offer him/her. The client finds out and constructs step by step what (s)he wants as the result of the service provider's reactions during the definition phase of the service. In fact, the service generation is a nondeterministic process (as we will see in section 4.2) depending on the conversation between two agents. As an example in current life, when somebody looks for clothes, *buying ready-to-wear clothes* is analogue to asking for a product, whereas *having clothes made by a tailor* is analogue to requiring a service to be generated.

Of course, the service generation approach does not aim to substitute product delivery. Most of the times, the latter gives satisfaction to the user. However, many scenarios such as, for example, planning a travel, looking for a job, learning a skill etc. cannot be modelled within this approach. One needs to adopt the dynamic service generation one. Actually, a product can be considered as the result of a one-shot interaction process between a pair: $\{$*customer, service provider*$\}$ while a service generation might be viewed as the result of the activation and management of a process defined by the triplet: $\{$*customer, service provider, conversational process*$\}$. In the former, the process is in principle deterministic, in the latter it is not: conversations are not a priori determined. Therefore, we need an open and dynamic communication model able to generate these conversational processes occurring between agents: AA-AA, AA-HA, and also HA-HA.

We will not go into more details about the generation of services now since we wanted, for the moment, to present the purpose of our work and the context of the paper: to build a model that represents a realistic progress toward the **dynamic service generation by conversational agents**.

### 1.4 Organisation of the paper

The proposed model is the result of research in three domains: i) the ITS or AI in Education, with its historical weight as a major source of inspirations for advances in computing and human learning, ii) agent communication, with ACLs (Agent Communication Languages) and their semantics, fundamental for conversation modelling, iii) language interpretation and applicative programming languages, in particular work on reflection and meta-evaluation. The goal of communication is certainly to change the interlocutor's state. This change is done after evaluating ($3^{rd}$ domain) new elements brought by the communication ($2^{nd}$ domain) in order for agents to learn ($1^{st}$ domain). We propose in this article a learning-by-being-told model that allows to carry out this change. The rest of the paper is organised as follows: Section 2 is a brief overview of the above three domains, presenting them and gluing them together. We also present here the foundations of our work (i.e. the STROBE model). Section 3 formalises our model, presenting agents as interpreters, introducing the concept of Cognitive Interpreter, and explaining how the representation of the other is realised. Section 4 presents some experimentation, simple, but significant, which aim to illustrate: i) the major feature of the model allowing in a quite simple way meta-level learning by communicating, ii) the utility of the model for dynamic service generation. A discussion on the potential effect of our choices with respect to the service generation concepts is presented in section 5. A conclusion, in section 6, rounds off the paper.

## 2 Agent communication, learning in MAS and language interpretation

### 2.1 Communication in MAS

Simply grouping together several agents is not enough to form a MAS. It is communication between these agents that makes it. Communication allows cooperation and coordination between agents [Fer99]. In the rest of the article we will deal with asynchronous direct mode communication. That means, a communication throws directly messages (that can be buffered) between agents.

Modelling communication has always been a problem. The traditional communication model of Shannon is not sufficient because in real communicative situations we can not consider communication as simple information transmission; we have to consider the whole of the dialogue. In order to model dialogue, we need first a communication language and a structure to model sets of communications, what is called conversations. Communication languages were firstly studied by Cohen, Perrault and Levesque [CP79, CL90] highly inspired from the speech act theory of Searle [Sea69] and Austin [Aus62]. These works on mental states, intention and performatives prefigured the apparition of agent architectures (or policy or theory of agency) based on mental states representation, such as BDI (Believe Desire Intention) and ACL such as KQML (Knowledge Query and Manipulation Language) [LF97]

and FIPA-ACL (Foundation for Intelligent Physical Agents - ACL) [Fip02]. Concerning conversation modelling the main approach in the agent community is the interaction protocol/conversation policies approach [GHB00]. These protocols represent the interaction structure and specify interactions that should be respected during conversation. A famous example of interaction protocol is the Contract Net protocol proposed by Smith. Interaction protocols are often represented with Finite State Machine or Petri Nets [CCF+00] and more recently with AUML, UML extension to agent.

From the agent communication literature we can extract two requirements that an agent communication model should provide, with respect to which our proposal is compliant:

- If we consider the communication effects on interlocutors, then we must consider that agents can change their goal or beliefs while communicating. Thus, they must be autonomous and should adapt during communication. Agents need a language to communicate assuming, as we do, that agents share a minimal common language, what is important is to allow language enrichment. Section 3.1 will explain how the concepts of Cognitive Environment/Interpreter may help us to dynamically modify and adapt our agent knowledge and behaviour by enriching the communication language at run time.

- Agents should interact with other agents (AA or HA) following the same principles, the agent interaction model should be generic with respect to the type of the communicating agents. What is important is the agent representation of each of its interlocutors, the partner model. Section 3.2 will detail how we deal with this point in our model.

## 2.2 The STROBE model

Our model can be viewed as the follow up of the STROBE model described in [Cer96b, Cer96a, Cer99]. STROBE shows how generic conversations may be described and implemented by means of *STReams* of messages to be exchanged by agents represented as *OBjects* exerting control by means of procedures (and continuations) and interpreting messages in multiple *Environments*. These three primitives are supposed to be first class objects[6] giving to the model an important dynamicity. The STROBE model is influenced by traditional ACLs such as KQML but goes beyond them as a consequence of lying on three main concepts adopted:

**Agents as interpreters.** While communicating, an agent executes a REPL (Read - Eval - Print - Listen) loop waiting for messages, selecting one, interpreting it and sending an answer. This loop is quite similar to the cycle of language interpreter. This principle is important because it regards agents as autonomous entities whose behaviour and interactions are controlled by a concrete evaluation procedure. You can notice that the role of the interpreter is both to interpret the message and its content. We will call `evaluate`, the procedure which emphasises the agent and interprets the message contents, and `evaluate-kqmlmsg`, the sub-procedure able to interpret messages. Notice that the choice of interpretation – as opposed to compilation and execution – is important in order to ensure the run time dynamicity of agents.

**Cognitive Environments.** Multiple environments are simultaneously available within the same agent. These environments represent the agent knowledge – for example, the value of a variable, the definition of a procedure – they embody the agent Knowledge Base (KB) as they represent the languages known by an agent. Actually, for the representation of each partner, STROBE proposes to have a partner model to be able to reconstruct as much as possible, the partner's internal state. It proposes to interpret each dialogue in a specific environment. Agents have two types of environments: i) the first one is unique – global and private – it belongs to the agent, representing its own beliefs, ii) the other ones – local – are dedicated to the interlocutors, each of them represents a partner model as it evolves during conversations. An agent can choose in which environment it can interpret a message. The concept of Cognitive Environments was firstly presented in [Cer96a, Cer96b] and we propose in section 3.2 to complete it.

[6] This notion is due to Strachey, first-class objects may be named by variables, may be passed as argument to procedures, may be returned as results and may be included in data structures.

**Streams processing.** STROBE messages are represented by streams. Streams lie on lazy evaluation mechanism (also called delayed evaluation) [ASS96]. Streams are ”lazy lists” (i.e. the natural representation of sequences that are partially instantiated (evaluated), during conversations). Agents used it to operate on incoming streams of messages, i.e. they generate the next message to send only after having received the last interlocutor answer. This allows to remove the necessity of global conversation planning (i.e. the classic interaction protocol) substituting it by history memorisation and one-step, opportunistic planning.

## 2.3 Learning in MAS

Learning and knowledge communication is very important in MAS, it provides evolution and adaptation of these agent societies through time. Learning was firstly limited to AI, but is nowadays extended to DAI and MAS. Here we will simply detail our overview of learning in order to bind it with the proposed model.

**Learning-by-being-told paradigm.** Considering agent learning, i.e. reflecting about what an agent learns on itself and on its interlocutors, we can, in a first approximation, distinguish three types of learning:

- The first one could be called *machine learning*, as for example reinforcement learning;
- The second one could be called *learning-by-being-told*. It comes from DAI and is issued from the instructional effect where an agent changes or creates representations according to received messages;
- The third one could be called *learning as a side effect of communication*. It is quite ”serendipitous”[7] as it occurs when an agent learns without being aware of it.

We will describe extensively here just an instance of the second type of learning. Our agents learn-by-being-told by other agents, they change their representations according to their interlocutors information given by messages.

**Levels of abstraction and learning.** Humans learn facts, rules (or procedures or skills), and languages necessary to understand messages stating facts or procedures, as well as necessary for generating behaviour when applying a particular procedure to arguments. In the same way, facts, rules and languages are such as *Data*, *Control* and *Interpreter* levels in computing. These three abstraction levels may be found in all programming languages. Indeed, an interpreter is the language evaluation expression mechanism, which defines a language; it is the interpretation process which gives a value to (evaluate) an expression. Every language provides an abstraction tool[8] which allows to enrich the language. For example, abstraction at the Data level consists in assigning values to already existing variables, or defining new data (i.e. in Scheme it is done with expressions such as `(define weekend 3)` or `(set! weekend '(friday saturday sunday))`). Abstraction at the Control level consists in defining new functions abstracting on the existing ones (i.e. in Scheme it is done with expressions such as `(define foo (lambda ...))`). However, the abstraction tool can usually reach only the first two levels. In order to abstract at the Interpreter level, what we call meta-level abstraction or meta-level learning, we have to modify the language interpreter itself. This feature is necessary in order, for instance, to add some special forms to the language. As an example, the special form `if` must be intrinsically implemented at the interpreter itself, it can not be defined at the Control level. Changing the Interpreter level must be done as well, in order to change the way of interpreting expressions, for example, in order to pass from an applicative order evaluation to a normal order evaluation, or to a lazy-evaluation.

The two first levels can be reached dynamically during execution: new data and procedures are added to the language each time an abstraction is interpreted. This is done by STROBE agents when modifying a dedicated Cognitive Environment. The challenge is to allow Interpreter level modifications at run time, in order to generate dynamic processes. It is what our model does. Our idea is to map these three levels schema into agents, considering them as set of pairs: environment + interpreter. While enabling its interpreter to evolve, an agent learns more than simple information, it completely changes its way of perceiving and processing this information. This is the difference between learning a datum and learning how to process a class of data.

[7] From serendipity: the faculty of making fortunate discoveries by accident.

[8] For example, the `define` special form in Scheme.

### 2.4 Scheme, and the influence of the applicative language community

To solve problems, we need tools that are not more complicated to use than the problem is to solve. The Scheme language is one of these tools. Scheme (Lisp-like language) is based on lambda calculus, hence it offers a clear semantics. Further, it is easy to learn and use [ASS96]. Scheme provides especially a powerful and simple memory model via first class environments as well as a very flexible and dynamic control model via first class procedures and first class continuations[9]. Scheme allows representing procedures as data easily enabling to write programs which manipulate other programs (i.e. interpreters or compilers). Streams, objects and environments are three Scheme first class primitives, which explain the choice of this language to embody the STROBE model. We also decided to use Scheme to reach, at run time, the Interpreter level. Scheme offers some devices to allow the dynamic modification of an interpreter[10], by means of:

1. Using a reflective interpreter with the mechanism of reifying procedures such as in [JF92];
2. Using first class interpreters such those present in [IJF92];
3. Using two levels of evaluation in order to apply/use the evaluation function (`eval`) which is part of the language and reachable.

## 3 The model

The extension of the STROBE model described in this paper can be synthetically described by one sentence: To augment the concept of Cognitive Environments, by including in them Cognitive Interpreters, which can be dynamically modified, allowing agents to develop a complete language (Interpreter + Environment) for each of its interlocutors, increasing agent dynamicity and giving to agents an evolutionary representation of the other.

### 3.1 Agents as a set of Cognitive Interpreters

Cognitive Environments in STROBE provide to agents a global environment and several local environments representing the others. Our extended model provides to agents not just one interpreter but several interpreters, including a global one and local ones for each agent they have a representation of. We therefore augment indeed the previous concept of Cognitive Environments. These interpreters (i.e. `evaluate` procedures) are stored in environments as any other usual procedures. We use the terms of: i) Cognitive Environments, because, according to the three learning levels seen above, the Data and Control levels can be modified by communications; and ii) Cognitive Interpreters, because the last level, the Interpreter level, can be also modified using devices presented in section 2.4. The three levels are reachable at run time while communicating, allowing agents to dynamically modify their meta-level knowledge increasing their autonomy and adaptability allowing them "to change their mind" i.e. their way of interpreting things. The model is formalised as:

$$AGENT_x = \{ENV_x^x + \{ENV_j^x\}^*\} \ \forall \text{j interlocutor}$$

$$ENV_j^i = \{INT_j^i + \{BIND_j^i\}^*\}$$

$$INT_j^i = \lambda : EXP \rightarrow evaluate_j^i(EXP, ENV_j^i)$$

$$BIND_j^i = \{(var, val)^*\}$$

$$\text{With, } ITEM_Y^X = \begin{cases} \text{X local item dedicated to Y} & \text{if} \quad X \neq Y \\ \text{X global item} & \text{if} \quad X = Y \end{cases}$$

An agent x, $AGENT_x$, is considered as set of environments, $ENV_j^x$, not null, because it unavoidably has its own global environment, $ENV_x^x$, and eventually a set of environments corresponding to is partner models. An environment, $ENV_j^x$, is a set of bindings with a least, an interpreter, $INT_j^x$, which is the Cognitive Interpreter included in the Cognitive Environment, and the rest of the bindings, $BIND_j^x$, which is a set of pairs variable – value. An interpreter, $INT_j^x$, is a procedure $evaluate_j^x$, which interpret an expression $EXP$, in the corresponding

[9] In programming languages a continuation is the next expression to evaluate with the result of the current evaluation.

[10] We experiment solution 1 and 3 in our model.

environment, $ENV_j^x$. In fact, our model enriches the STROBE model by adding to it a new first class primitive: the Interpreter[11].

## 3.2 Representation of the others

In our model, the Cognitive Environment and its included Cognitive Interpreter, emphasise the partner model. Therefore, message interpretation is done in a given environment and with a given interpreter both dedicated to the conversation (we deal with conversations because Cognitive Environment modifications[12] are persistent along communications). For instance, Figure 1 points out the representation of an agent, $AGENT_A$ which know two others agents, $AGENT_B$ and $AGENT_C$.

$$AGENT_A = \begin{cases} ENV_A^A = \begin{cases} INT_A^A = \lambda : EXP \rightarrow evaluate_A^A(EXP, ENV_A^A) \\ BIND_A^A = \{(u,3)(v,5)(square, \lambda : x \rightarrow x * x)\} \end{cases} \\ ENV_B^A = \begin{cases} INT_B^A = \lambda : EXP \rightarrow evaluate_B^A(EXP, ENV_B^A) \\ BIND_B^A = \{(u,3)(v,5)(square, \lambda : x \rightarrow x * x)\} \end{cases} \\ ENV_C^A = \begin{cases} INT_C^A = \lambda : EXP \rightarrow evaluate_C^A(EXP, ENV_C^A) \\ BIND_C^A = \{(u,6)(v,2)(id, \lambda : x \rightarrow x)\} \end{cases} \end{cases}$$

Figure 1: Representations of $AGENT_A$, with two partner, $AGENT_B$ and $AGENT_C$, models

When an agent meets another one for the first time, three cases are possible: i) First, it produces a new environment for this agent by cloning its own environment and using it to communicate with this agent. For example, when two agents meet together for the very first time, and when they do not have any idea of the type of this new agent (e.g. HA, AA, service provider, etc...). ii) Secondly, it produces a new environment for this agent by cloning another agent environment it already has. For example, when another agent finishes a conversation (e.g. generate a service) that a first one has begun. iii) And thirdly, it uses an already existing environment to communicate with this new agent. For example, when an agent identical, or playing the same role, or belonging to the same group, replaces another one. The third situation is heavy in consequences, because it shows that several agents can be represented for another one by the same partner model. Therefore, an agent may have, representation of other lonely agents, but also representations of groups of agents eventually playing different roles. It brings our model closer to an organisational structure as it is proposed by Singh in [Sin98].

It is important to notice that the global environment, (i.e. $ENV_x^x$), is private and changes only for precise reasons. However, caring readers could have noted that in Figure 1, $BIND_C^A$ was different from $BIND_B^A$ and particularly from $BIND_A^A$. Which value $AGENT_A$ must use for *u* or *v*? When $BIND_A^A$ must change? It will be the subject of discussion in section 5.

Let us come back to the REPL loop proposed by STROBE that we still consider. This REPL behaviour is illustrated by Figure 2. Each time an agent *Read*s a message, it picks up both the dedicated environment and interpreter to interpret it, then it *Eval*s it (i.e. apply the selected interpreter on the message in the selected environment, and *Print*s the corresponding answer (i.e. send the answer message and/or process the evaluation result), before *Listen*ing the next message.

## 3.3 Communication language

In this section we go into details of the communication language we use (i.e. message structure and used performatives). This communication language is considered minimal. Messages we use are speech act oriented and

[11] It is a first class object because an interpreter is a procedure and Scheme procedures are first class.

[12] We will generally now use the term of Cognitive Environment, or simply environment, but the reader should not forget that it concerns also Cognitive Interpreters due to the fact that the latter is included in the former.

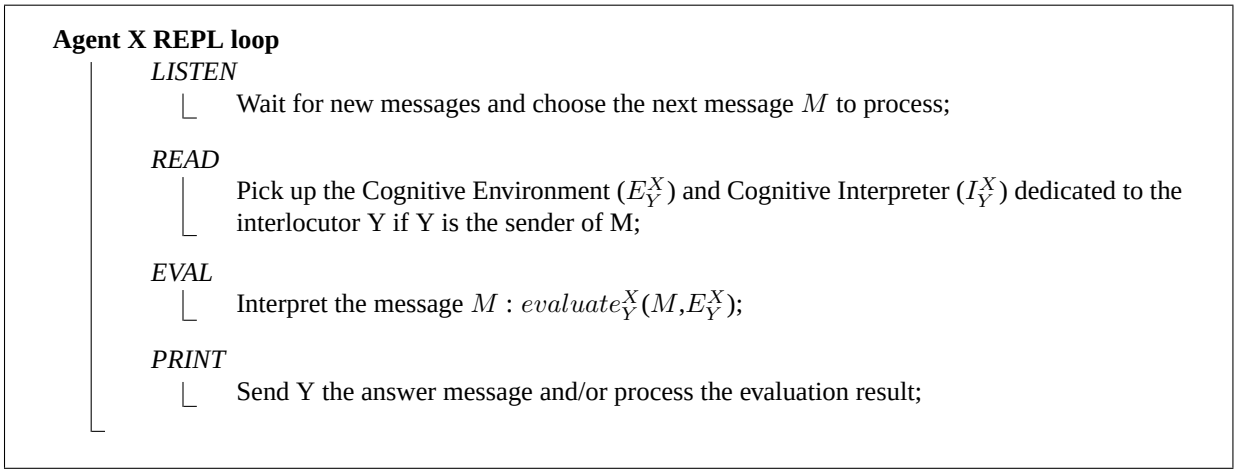


Figure 2: Agent REPL loop

use performatives. They are inspired by the KQML or FIPA-ACL message structure but use only the primordial parameters; other ones (e.g. ontology, in-reply-to, reply-with, protocol) may easily be added. Their form is:

$$MSG = \{AGENTs, AGENTr, PERFORM, CONTENT\}$$

with $PERFORM = \{assertion, ack, request, answer, order, executed, broadcast\}$

The sender ($AGENTs$) and the receiver ($AGENTr$), are the two agents concerned by the message. The $CONTENT$ of the message is an expression written in a content language. Our model uses the same language both for the message and its content (i.e. Scheme) as we can use the same interpreter to evaluate the message and its content. In order to construct a minimal agent we propose to process six performatives by default, five presented in [Cer99] (Table 2) and the latest one, broadcast, that is the subject of one of the experimentations hereafter:

**assertion** messages modify interlocutor behaviour or some of its representations. Answers are **ack** (acknowledgement) messages reporting a success or an error. For example, in Scheme, content of assertion messages are `define` or `set!` expressions. These messages are used to perform classical Data/Control levels learning.

**request** messages ask for one interlocutor representation, such as the value of a variable or the closure of a function. Its **answer** returns a value or an error. In Scheme, request messages are certainly variables names.

**order** messages require the interlocutor to apply a procedure. This interlocutor sends the result as the content of an **executed** message. In Scheme, it would correspond to a procedure call, `(foo ...)`. These messages are used, for instance, to dynamically modify an agent interpreter.

**broadcast** messages consist in sending a message with a pair as content `(perform, content)` that means that the interlocutor must send a message with the performative `perform` and with the content `content` to all is current interlocutors. There is no answer defined for the broadcast messages.

# 4 Experimentations: examples of scenarios

In order to express the potentiality of our model, we propose in this section two examples of scenarios, called experimentations, which could occur with such a model. They illustrate: i) Meta-level learning by communicating: agents which modify their interpreters while communicating. ii) How our model enables the dynamic specification of a problem by means of communication, which is necessary for service generation as presented in section 1.3. Even if these experimentations are ”toy examples”, they are significant as they represent a solution for a large class of problems.

## 4.1 Meta-level learning by communicating

**A teacher-student dialogue.** The first experimentation shows how an agent can learn-by-being-told a new performative, thus dynamically modifies its message interpretation procedure (i.e. `evaluate-kqmlmsg`). It is a standard "teacher-student" dialogue. An agent teacher ($A_T$) asks to an agent student ($A_S$) to broadcast a message to all its interlocutors using a special performative, `broadcast`. However, student does not initially know the performative used by teacher. So, teacher transmits two messages (`assertion` and `order`) clarifying to the student the way of processing this new performative. Finally, teacher formulates once more its request to student and obtains satisfaction. After the last message process, $evaluate-kqmlmsg_T^S$ procedure is modified. Thus a part of its interpreter ($INT_T^S$) was dynamically changed and the corresponding definition in its dedicated environment ($ENV_T^S$) is changed. It is thus a meta-level learning. Then student agent can process `broadcast` messages. Figure 3 describes the exact dialogue occurring in the experimentation. The paper [JC03] details this experimentation and shows it using the reifying procedures mechanism cited in previous section.

| Teacher ($A_T$) | Student ($A_S$) |
|---|---|
| *Here is the definition of the square procedure:*<br>$\{A_T, A_S, assertion,$<br>`(define square (lambda (x) (* x x)))`$\}$ | *Ok, I know now this procedure:*<br>$\{A_S, A_T, ack,$<br>`(*.*)`$\}$ |
| *Broadcast to all your current interlocutors:*<br>$\{A_T, A_S, broadcast,$<br>`'(order (square 3))`$\}$ | *Sorry, I don't know this performative:*<br>$\{A_S, A_T, answer,$<br>`'(not-such-performative broadcast)`$\}$ |
| *Ok, here is the method to add this performative to those you know:*<br><br>*Here is the code you have to generate and add to your* `evaluate-kqmlmsg` *procedure:*<br>$\{A_T, A_S, assertion,$<br>`broadcast-code`$\}$<br><br>*Run this procedure:*<br>$\{A_T, A_S, order,$<br>`(set! evaluate-kqmlmsg with-broadcast)`$\}$ | *Ok, I have added this code in a binding of my environment:*<br><br>$\{A_S, A_T, ack,$<br>`(*.*)`$\}$<br><br>*Ok, I have just modified my interpreter:*<br>$\{A_S, A_T, executed,$<br>`(*.*)`$\}$ |
| *Broadcast to all your current correspondents:*<br>$\{A_T, A_S, broadcast,$<br>`'(order (square 3))`$\}$ | *Ok, I broadcast*<br>$\{A_S, \ldots, order,$<br>`(square 3)`$\}$ |

Figure 3: `broadcast` learning teacher-student dialogue.

Remark — Note that `broadcast-code` variable contains the code that $A_S$ must integrate to its procedure `evaluate-kqmlmsg`. The term integration is important, because $A_S$ does not simply replace its procedure by the $A_T$ one, it integrates this code to its previous procedure yet conserving as valid the previous modifications on it. This is thus a constructivist method. Generated code is stored in a lambda-expression (i.e. procedure in Scheme) called `with-broadcast` and after the assignment (`set!`) message evaluation, this lambda-expression becomes `evaluate-kqmlmsg`.

## 4.2 Enabling dynamic specification by communication

If we consider the agents as interpreters, then we can consider them as nondeterministic interpreters. The key idea is that expressions, in a nondeterministic language, can have more than one possible value. With nondeterministic evaluation, an expression represents the exploration of a set of possible worlds, each determined by a set of choices. Nondeterministic evaluation is very useful for Constraint Style Programming (CSP). We invite the reader to refer to [ASS96] (chapter 4) for more details on nondeterministic evaluation.

**An e-commerce scenario.** It is interesting for our agents to be considered as nondeterministic interpreter to solve, for instance constraint based programs. But above all, the most interesting thing is that our agents can progressively build such programs while communicating with other agents and then apply these programs to generate a solution (to achieve a task) for another agent. The constraints, defining a nondeterministic program, can be determined

progressively with the conversation using our model to dynamically change procedures and the way of interpreting them. Let us consider, a standard e-commerce dialogue for a train ticket booking. A ticket is characterised by a departure city, a destination city, a price, and a date. A SNCF[13] agent dialogues with a customer agent which tries to book a ticket. A realistic dialogue could be:

**a.** Customer: *I want a ticket from Montpellier to Paris*

**b.** SNCF: *Ok, what are your conditions?*

**c.** Customer: *Tomorrow before 10AM. Please, give me a proposition for a ticket!*

**d.** SNCF: *Ok, train 34, departure tomorrow 9.30AM, from Montpellier to Paris, 150€*

**e.** Customer: *Is it possible to pay less than 100€?*

**f.** SNCF: *Ok, train 31, departure tomorrow 8.40AM, from Montpellier to Paris, 95€*

**g.** Customer: *Another proposition please?*

**h.** SNCF *Ok, train 32, departure tomorrow 9.15AM, from Montpellier to Paris, 98€*

**i.** Customer: *Ok, I accept this one*

We can notice that the interactions **a**, **b**, **c** and **e** deal with the constraints on ticket selection. The interactions **d**, **f** and **h** are result of application of ticket research function, with various constraints. The interaction **g** corresponds to a request of the customer to get another answer, that means to explore another branch of the solution tree. Figure 4 illustrates how this dialogue can be represented into Scheme expressions in order to be realised by our agents.

The customer agent ($A_C$), which represents an HA as the user of the service, transmits its requests with `require` and `try-again` expressions. At the beginning of the conversation, SNCF agent ($A_S$) starts a new `find-ticket` function construction which is dynamically modified and built progressively during the conversation. These modifications consist in changing a value in the SNCF agent environment dedicated to the customer agent, $E_C^S$. This idea is interesting because it is the dialogue that builds the computation to be carried out and not the opposite. It is a typical scenario that we could find in many e-commerce applications or other ones of the same kind where agents must build a program to find a solution together. The classical approach of program construction (which can be found in traditional software engineering) that specifies the problem before coding it, is changed into a dynamic specification approach during coding. That is to say, specification and realisation are, with our model, carried out at the same time (in one step).

The above constraints are quite simple ones, but with such a system we can have constraints as complex as: conjunction and disjunction of predicates, or conditionals which bind constraints together. Therefore, this approach enables the user to explore a large possibility in its own constraint definition. In fact, in this experimentation $A_S$ does not want to know values of variables (i.e. departure city, price, etc...), it simply exposes to $A_C$ the variables on which it can express constraints, and it allow it to communicate its ideas. The travel agency example is often used in Web research community (Semantic Web, ontology building, Data mining etc...), but seldom associated to a dynamic interaction with the user as our approach presents.

# 5 Discussion

**About the experimentations.** The first experimentation shows how to add a new performative to the ones already known by an agent, thus how to modify an agent messages interpretation procedure (i.e. `evaluate-kqmlmsg`). The same principles can be used to modify any part of an agent interpreter. We can imagine a prelude to the second experimentation with a scenario where an agent teaches to another one how to transform its interpreter into a nondeterministic one. With this model, our agents have a set of environments and interpreters that represent their knowledge and its evolution through time. Indeed, these pairs correspond to their recognised sub-languages and thus to their faculties to carry out a task. Agents can process programs for others or even exchange their interpreters just like in Grid architectures where it is more interesting to move program than data. Their interpreters can also be transmitted before a conversation, just like an ontology.

[13] Due to the French acronym Société Nationale des Chemins de Fer.

| Customer | SNCF |
|---|---|
| *I want a ticket from Montpellier to Paris*<br>`(require`<br>`(eq? depart 'montpellier))`<br>`(require`<br>`(eq? dest 'paris))` | Definition of a new `find-ticket` function:<br>`(define (find-ticket)`<br>`(let ((depart (amb *city-set*))`<br>`(dest (amb *city-set*))`<br>`(price (amb *price-set*))`<br>`(date (amb *date-set*)))`<br>`(require (not (eq? depart dest)))`<br>`(require (eq? depart 'montpellier))`<br>`(require (eq? dest 'paris))`<br>`(list (list 'depart depart)`<br>`(list 'dest dest)`<br>`(list 'price price)`<br>`(list 'date date))))`<br><br>*Ok, what are your conditions?* |
| *Tomorrow before 10AM*<br><br>`(require (< date *tomorrow10AM*))`<br>*Please, give me a proposition for a ticket!*<br><br>`(find-ticket)` | `find-ticket` fonction modification adding a new constraint. Then procedure execution:<br><br>*Ok,train 34, departure tomorrow 9.30AM, from Montpellier to Paris, 150€*<br>`((depart montpellier) (dest paris) (price 150)`<br>`(date *tomorrow9.30AM*))` |
| *Is it possible to pay less than 100€?*<br>`(require (< prix 100))`<br><br>`(find-ticket)` | idem<br>*Ok, train 31, departure tomorrow 8.40AM, from Montpellier to Paris, 95€*<br>`((depart montpellier) (dest paris) (price 95)`<br>`(date *tomorrow8.41AM*))` |
| *Another proposition please?*<br>`(try-again)` | `find-ticket` function execution:<br>*Ok, train 32, departure tomorrow 9.15AM, from Montpellier to Paris, 98€*<br>`((depart montpellier) (dest paris) (price 98)`<br>`(date *tomorrow9.15AM*))` |
| *Ok, I accept this one* | |

Figure 4: Scheme expressions in dialogue between SNCF agent and customer agent.

**Learning on Cognitive Environments.** Even if our model is an instance of the learning-by-being-told paradigm, we may consider an internal learning on the different Cognitive Environments. Learning of the first type as it was presented in 2.3. For instance, the following two processes:

- The first one consists on learning on a particular environment to deduce new knowledge;
- The second learning process concerns the agent global environment, $ENV_x^x$. We have to consider that agents may transfer the knowledge they learned in the different $ENV_i^x \; \forall i \neq x$ (local environments) to $ENV_x^x$ (the global environment). They have to do that in order to take into account what they have learned in a global way. It is part of their evolution. They definitively have to do that if they want to reuse the knowledge learned within different conversations.

**Towards dynamic service generation.** In some aspects, service generation is analogue to information retrieval with the following properties: i) The database is so large that HA cannot specify a single, simple database query for which there is one single answer; ii) HA may not necessarily have a definitive set of constraints in mind when starting the search task, or may change constraints during the message exchange, and iii) There may be multiple search goals. We believe that service generation will help in Web scenarios such as those related to information retrieval. The service generation approach does not aim to replace product delivering but rather to improve and enhance it. Generating a service means understand and construct, by interactions what a user (an agent) needs. If a user has precise needs or precise ideas of what (s)he wants the system to give her/him, then interactions are simple

and a ”product” may be rapidly delivered. Actually, a system which can generate services, can easily deliver products. As in section 4.2, asking constraints on variables include asking value on these variables, but not the opposite. For example, in the e-commerce experimentation, constraints expressed by the customer agent can be equalities (i.e. values) for instance: `(require (= price 95))` or `(require (= date *tomorrow10AM*))` etc. This is equivalent to fill a Web form what is a current practice in many Web sites. One requirement of service generation is that the user does not certainly know what he wants. **This statement supports our view that service generation implies learning**. At the end of the process the user has certainly learned something: in the e-commerce experimentation, the user, communicating with the customer agent, has simply learned what will be the train he is going to take. Even if the e-commerce scenario is a very simple one, it may be easily extended to complex ones and therefore imply very complex learning. In this sense, our work is inscribed in a global logic to provide an e-learning infrastructure based on dynamic service generation. **That means that an agent able to generate dynamically services embody perfectly Learning Services. As our agents evolve on the Grid, they can realise dynamic generation of Grid Learning Services**.

**Grid Learning Services.** We take the challenge of Semantic Grids by interpreting the role of Grids as societies of autonomous interacting agents offering dynamically defined as well as dynamically delivered (generated) services by means of conversations among AA and HA available in the society. This view is not only consistent with current developments on Grid, but with the above outlined state of the art in agents and agent communication, and with our own extension to include HA. The feasibility of the approach is ensured by our experimentations, in the sense that they represent a quite comprehensive class of potential scenarios on future Grids. Learning Grids (i.e. societies of learning agents) become in this view at the same time societies of agents (HA and AA) supporting human learning. No doubt that HA are autonomous (what is yet to be realised for AA) and solve problems by interacting (what is partially realised by AA). No doubt that their own ”internals” are hard to identify (what calls for a social, better than a mental state based, approach to modelling cooperation in mixed societies of agents).

In the paper we have outlined how far we can go at the moment in modelling and constructing autonomous, interacting and socially committed AA. The collaboration among the two classes (AA and HA) is based on a clear commitment on the communication language, that for the moment may be quite limited in its expressiveness but may become very powerful if the service offering semantics become a practice on the Web and on the Grid. One fundamental requirement for a progress in the area is conceptual simplicity for the researcher/developer. We are all aware that the increase in complexity of current proposals around the Grid as well as in the agent world is uncontrolled, therefore quite threatening. We think that our proposed model is perceived as simple enough to be not only understood in its potential, but also used and reused in concrete settings, as we are convinced that it may be the kernel of complex applications, by composition, abstraction and generalisation, yet maintaining its simple foundations.

# 6 Conclusion

In this paper we try to convince e-learning community that improving AA-learning improves also HA-learning. Therefore, we have presented our model of communication and representation of agents. We have thoroughly adopted the two requirements of section 2.1: the model i) is based on a strong representation of the others (thanks to the concept of Cognitive Environments) and ii) is highly dynamic, allowing the three levels of learning (Data, Control, Interpreter) at run time (thanks to the concept of Cognitive Interpreters). Our agents are able to interpret communication messages in a given environment, including an interpreter, dedicated to the current conversation. We show how communication enables to dynamically change values in an environment and especially how these interpreters, which represent the agents, can dynamically adapt their way of interpreting messages. Then, how our agents can build, dynamically while communicating, languages dedicated to their interlocutors. Besides, the model was illustrated with experimentations showing its adequateness and the feasibility of its prototypical applications. The model is also pertinent because it can embody classical approaches in agent communication. Indeed, you can have a message evaluation function which follows a special interaction protocol using a classical ACL such as FIPA-ACL.

This model is the result of research in different domains and therefore has the advantage of gluing them together in a coherent way. Working on agents, the Web, applicative programming languages, technologies supporting human learning and more recently on Grid, showed us **the importance of considering interactions as the core concept in future Social Informatics applications**. A strong interaction based system might be the only way to

integrate humans, then users, in the loop. Nowadays, interactions seem to become a real issue in computer science. Actually, we do not really communicate with our computers and they do not communicate together, we simply ask them to answer some precise questions, to compute some algorithms! However, as Turing himself said, Turing machines (i.e. nowadays computers) could not provide in principle a complete model for all forms of computation. Currently some researchers [Weg97]try to find out the way to shift from classical Turing Machines to Interaction Machines.

We are now facing new kinds of problems that only interaction can help to resolve. For us, a strong interaction based Grid, sharing any kind of resources, seems to be the key issue for the future, under the hypotheses that one is so modest to accept that interaction is indeed a new fundamental concept to be explored and modelled, and that any model (or experimentation, or language application) should try to remain simple enough to allow understanding and mastering of the complexity in terms of composition of simpler parts.

# 7 Annex

The presented model has been partially implemented in a prototype developed with MIT Scheme. The above experimentations have been tested and validated. The Scheme files and information about the model are available on http://www.lirmm.fr/~ jonquet.

# 8 Acknowledgements

Work partially supported by the European Community under the Innovation Society Technologies (IST) programme of the $6^{th}$ Framework Programme for RTD - project ELeGI, contract IST-002205. This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.

# References

[ASS96] Harold Abelson, Geral Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Massachusetts, USA, $2^{nd}$ edition, 1996.

[Aus62] John L. Austin. *How to do things with words*. Clarendon Press, Oxford, UK, 1962.

[CCF+00] R. Scott Cost, Ye Chen, Tim Finin, Yannis Labrou, and Yun Peng. Using colored petri nets for conversation model. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Artificial Intelligence*, pages 178–192. Springer-Verlag, Berlin Heidelberg New York, 2000.

[Cer96a] Stefano A. Cerri. Cognitive Environments in the STROBE model. In *European Conference in Artificial Intelligence and Education, EuroAIED'96*, Lisbon, Portugal, 1996.

[Cer96b] Stefano A. Cerri. Computational mathetics tool kit: architecture's for dialogue. In C. Frasson, G. Gauthier, and A. Lesgold, editors, *Intelligent Tutoring Systems, $3^{rd}$ International Conference ITS'96*, volume 1086 of *Lecture Notes in Computer Science*, pages 343–352. Springer-Verlag, Berlin Heidelberg New York, 1996.

[Cer99] Stefano A. Cerri. Shifting the focus from control to communication: the STReams OBjects Environments model of communicating agents. In Padget J.A., editor, *Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing*, volume 1624 of *Lecture Note in Artificial Intelligence*, pages 74–101. Springer-Verlag, Berlin Heidelberg New York, 1999.

[CL90] Philip R. Cohen and Hector J. Levesque. Intentions is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[CP79] Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[CSCM00] Stefano A. Cerri, Jean Sallantin, Emmanuel Castro, and Daniele Maraschi. Steps towards C+C: A language for interactions. In S. A. Cerri and D. Dochev, editors, *Artificial Intelligence: Methodology, Systems, and Applications, AIMSA'00*, volume 1904 of *Lecture Notes in Artificial Intelligence*, pages 34–48. Springer-Verlag, Berlin Heidelberg New York, 2000.

[EKD03] Marc Eisenstadt, Jiri Komzak, and Martin Dzbor. Instant messaging + maps = powerful collaboration tools for distance learning. In *TelEduc'03*, Havana, Cuba, May 2003. http://www.buddyspace.org.

[Fer99] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley Longman, Harlow, UK, 1999.

[Fip02] Foundation for Intelligent Physical Agents, FIPA ACL message structure specification, December 2002. www.fipa.org/specs/fipa00061/.

[FK99] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[FKNT02] Ian Foster, Carl Kesselman, Jeff Nick, and Steve Tuecke. The physiology of the Grid: An Open Grid Services Architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*. The Globus Alliance, June 2002. Extended version of Grid Services for Distributed System Integration.

[FKT01] Ian Foster, Carl Kesselman, and Steve Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Supercomputer Applications*, 15(3), 2001.

[GHB00] Mark Greaves, Heather Holmback, and Jeffrey Bradshaw. What is a conversation policy? In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Artificial Intelligence*, pages 118–131. Springer-Verlag, Berlin Heidelberg New York, 2000.

[IJF92] John W. Simmons II, Stanley Jefferson, and Daniel P. Friedman. Language extension via first-class interpreters. Technical Report TR362, Indiana University, Bloomington, Indiana, USA, September 1992.

[JC03] Clement Jonquet and Stefano A. Cerri. Cognitive agents learning by communicating. In *Colloque Agents Logiciels, Coopération, Apprentissage et Activité Humaine, ALCAA'03*, pages 29–39, Bayonne, September 2003.

[JF92] Stanley Jefferson and Daniel P. Friedman. A Simple Reflective Interpreter. In *International Workshop on Reflection and Meta-level architecture, IMSA'92*, Tokyo, Japan, November 1992.

[LF97] Yannis Labrou and Tim Finin. A proposal for a new KQML specification. Technical report TR-CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore, Maryland, USA, February 1997. www.cs.umbc.edu/kqml/.

[RJS01] David De Roure, Nicholas Jennings, and Nigel Shadbolt. Research agenda for the semantic Grid: A future e-science infrastructure. Technical report, University of Southampton, UK, June 2001. Report commissioned for EPSRC/DTI Core e-Science Programme.

[Sea69] John Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, Cambridge, UK, 1969.

[Sin98] Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.