Programming communicating systems and Lisp

Zoran Aleksić
Vinča Institute *
P.O. Box 522, 11001 Belgrade, Yugoslavia
zaleksic@acm.org

Abstract

Good way to explore the space of interacting programs more complex than current client/server based programs is to use Lisp with the embedded Elephant-like language and bottom-up programming style. There are indications that we can develop interacting programs that can exhibit patterns of communication similar to some patterns of human communication without using artificial intelligence. Features of Elephant-like programming languages are described.

1 Introduction

The explosive growth of computer networks and availability of fast computers have opened the space for the development of complex interacting programs. When we order interacting programs according to the complexity of their behavior, we find client/server based programs on the side of lower complexity. In the domain of interacting programs with high complexity we find artificial intelligence (AI) based programs like software agents. As Lisp users know, Lisp is not only good language for developing AI programs but it is a powerful language for writing advanced client/server based programs. For example, Common Lisp HTTP server written by Mallery [7] in relatively short time offers more flexibility and features than servers written in other programming languages.

Between the client-server based interacting programs and AI based programs, I expect we will find a large group of programs that can interact in artificial languages which have many features of natural languages. These programs won't require the use of AI for their functioning. I will describe these programs indirectly by explaining their features and describe their main building blocks that can be efficiently implemented using Lisp. The ideas I will present are build up on the McCarthy's proposal for Elephant 2000 programming language [12]. I have also taken some ideas from several, I believe, promising theories of communication and natural languages.

McCarthy uses Elephant 2000 as a vehicle for ideas about features of modern programming languages. Initially the Elephant language was connected with the novel approach to proving correctness of Lisp programs. Latter, the design of the language was combined and elaborated with the ideas about speech acts. Elephant language is intended for writing programs that interact with other programs and people. McCarthy's proposal is still in the draft form but it is quite clear specially when seen against the background of his other writings.

Several theories of natural language [9, 17, 8] that I consider useful for developing communicating programs/systems[1] are kind of bottom-up theories. They do not assume the existence of high level entities like thoughts or plans, but start from the stimulus-response elements of interaction and show how phenomena like speaking or meaning emerge in the dynamics of interactions. The difficulty of understanding the phenomenon of human communication lies in the necessity of analyzing it at different levels of hierarchy. At the level of social interactions we describe communication in terms of actions of coordinating common actions. At the level of messages we describe the grammar of the lan-

*Address in Japan: 3-3-20 Minamigaoka, Nisshin-shi, JAPAN 470-0114

[1] I expect that many communicating programs will be stand-alone systems in the future.

guage. At the level of biology we describe the triggering of neuronal activity as the generating base of communication. We do not understand human communication in details but it appears that at the various levels of analysis we find elements that have mechanical explanation. If the hypothesis that the meaning of words is something that emerges in the networks of social interactions is correct, we do not have to wait for the discovery of the principles of AI to develop interesting communicating systems.

In the following sections I will describe features of the communicating programs that can be efficiently written using Lisp:

- reactive,
- communicating by speech acts,
- performing communicative action,
- syntactically based computing,
- using the context of interaction,
- keeping the traces and effects of interaction,
- changing in the course of interaction.

Several theories of natural language[8, 15, 4, 3, 16] consider one or few features from this list as the fundamental features of human communication. Programs having these features can provide complex patterns of interractions. Notice that we don't require a communication language with the surface structure of a natural language as the feature of of communicating programs. Our goal is not to provide a system for natural language understanding. However, it is plausible that more complex communicating systems that imitate the surface structure of natural languages will exhibit strong Eliza effect [18].

An elegant way to write communicating programs would be to use Lisp extended with Elephant-like language. Embedding Elephant language in Lisp is straightforward. However, understanding the behavior of the implemented system could be difficult. Communicating programs written in Lisp with Elephant-like embedded language could be instances of complex systems. They could be simple programs exhibiting complex nontrivial behavior.

2 Listen-Evaluate-Respond Loop

At the lowest level of analysis the process of communication is the chain of mutual perturbation of two or more interacting systems. Linguistic behavior of the system A causes the the linguistic behavior of the system B that in turn becomes the stimulus for another linguistic behavior of the system A. Elephant 2000 programs can be also regarded as stimulus-response systems.

An Elephant program is the kind of rule-based reactive systems. Elephant language expressions are sugar coated logical sentences. Some sentences of the program are rules relating incoming messages with the responses. At the level of the runtime system it is provided that only one syntactically acceptable message reaches the program at a time.

For example, a program for airline reservation may receive a request for booking a seat for passenger "Bob" at the flight JL5:

request make commitment $admit("Bob", JL5)$.

The run-time system of the program provides the time at which the event of communication happens and the identity of the sender. The message is matched against the set of rules. The query can be replied triggering the rules

if $\neg full\ flt$
then accept.request
make commitment $admit(psgr, flt)$,

and

$full\ flt \equiv card\{psgr \mid$ **exists commitment** $admit(psgr, flt)\} = capacity\ flt.$

The identifiers in bold face are the actions provided by the language. Parenthesis are omitted for one argument functions and predicates. The action **accept.request** is a function taking one argument. It means 'do what is requested'. The function **commitment** generates an abstract object to be used latter in the programs actions. Internal action **make** changes the content of the program's data base. The identifiers flt and $psgr$ are names

of logical variables. The function $admit(psgr, flt)$ is defined in this particular program. It means 'admit the passenger $psgr$ to the flight flt'.

I am skipping explanation of other details and I won't show other programs statements. I only want to illustrate that the the arrival of the message triggers some computations and causes some changes in the program data-base.

Pattern matching can be done with one of many algorithms developed in Lisp. The matching procedure doesn't have to be more complex than the one used in Prolog.

In the simplest implementation we can have a loop checking if a message has arrived. When the message arrives appropriate rule is selected and the relevant computation is activated. Better and more interesting implementation would exploit multiprocessing available (but not yet standardized) in many Lisp implementations.

3 Speech acts

In the previous section I considered communication at the operational level. Let me emphasize again that human communication is a complex process that can be described at different levels of analysis. At the pragmatic level, we can consider the aims of communication. One of the descriptions of human communication at this level of analysis is the speech act theory.

Speech act theory has been developed to overcome the narrow focusing on the referential use of language [15]. Conversation is fundamentally a social activity. In saying something we do not only describe things and events but also we can perform other actions like making commitments or influencing the thoughts, feelings and actions of others. In general, speaking is a kind of purposeful social activity in which we refer to something, express ourselves, create states of affairs and social relationships.

In speech act theory the minimal unit of human communication is called the illocutionary act. One component of illocutionary act is the act of the speaker uttering a sentence. When an illocutionary act is successfully performed the hearer will always reach the understanding of the utterance. In addition to the illocutionary effect of understanding, utterances may produce further effects on the feelings, and attitude of the hearer. The acts causing these effects are called perlocutionary acts, and the effects perlocutionary effects. The perlocutionary effects can be intended or are the side effects of the execution of illocutionary act.

Speech act theory is focused on determining the details of illocutionary act like what conditions have to be satisfied to have a successful performance of speech act. Assuming that the speakers are sincere, that is, they mean what they say, perlocutory acts can be considered as derived from illocutionary act.

The sentences of speech acts can be thought of as having two components: propositional content and the illocutionary force. The propositional component can be interpreted as a denotational expression. Illocutionary force is the implicit or explicit component that determines how the propositional content is to be used.

Elephant programs communicate using speech acts. McCarthy proposes to provide speech act like requests or promises. Sentences of Elephant speech act would be in Elephant I-O language whose meaning would be partially determined by the Elephant language and partially determined by the program itself.

Illocutionary component is explicit in sentences of Elephant I-O language. For example, in the composite sentence

request make $alfa$,

the illocutionary component **request make** is combined with the propositional component labeled as $alfa$. Like the exaple in the second section we can take $alfa$ = **commitment** $admit("Bob", JL5)$. We can change the illocutionary component leaving the propositional component invariant. The expression **query exist** $alfa$ has the straightforward meaning. Both illocutionary component and propositional component can be composite expressions.

Notice, that the illocutionary component of the current speech act can become the subject of the propositional component of one of following speech acts. This explicit separation in time makes simple and safe to have self-referential communications.

Speech acts are used in the languages for agent communication KQML (knowledge query and ma-

nipulation language[2]) and ACL (agent communication language) [6]. These languages are quite simple. They are only used for expressing the illocutionary component of the message. The choice of the language of the propositional component (content) is left to the designer/implementer of the agent.

4 Communicative action

Speech act theory models the structure of conversation as a network of illocutionary acts. Each illocutionary act creates the possibility of a finite and usually quite limited set of illocutionary acts as replies. Having the small set of possibilities facilitates the implementation of speech-act theory. However, the theory focuses on single illocutionary acts and doesn't give enough clues on how to organize longer threads of conversation.

Habermas proposed the variant of speech act theory which he uses in his theory of communicative action [4]. The theory of communicative action is concerned with the coordination of common action of two or more persons. Participants of the common action coordinate their activity communicating using illocutionary acts.

Instead of having several types of illocutionary acts with different conditions of satisfaction, Habermas proposes only one type of illocutionary act. He calls it the standard speech act. The aim of standard speech act is to reach understanding about something in the objective, the social, and the subjective worlds. With every speech act the speaker implicitly raises three validity claims. She claims that propositional content of the utterance is true. She claims the rightness or appropriateness of norms of the potential interpersonal relation. She also claims that she is sincere and not having hidden agendas. An illocutionary act succeeds when the hearer comprehend the utterance and takes the interpersonal relation intended by the speaker. Accepting a speech act offer the hearer undertakes the obligations relevant for the sequel of interaction. Her acceptance is grounded on the assumption that the speaker is a responsible participant guaranteeing the conditions of acceptability of speech act offer.

[2]KQML uses Lisp syntax.

Some ideas of Habermas can be used to provide a new feature of Elephant language [1]. Habermas consider as a unit of communication a pair of speech acts, where the speaker makes an speech act offer and the hearer states her acceptance with yes or no. It is straightforward to extend this model to a sequence of illocutionary acts starting with the speech act offer and ending with the the speech act declaring whether the agreement is reached or not.

The objective, social, and subjective worlds of interacting programs are the sets of facts that differ according to the the criteria of acceptability. The subjective facts refer to the internal states of the interacting programs. The social facts are about the protocols and concrete common activity. The objective facts are sheared between different programs. All facts are propositionally structured. The agreement that can be reached between two or more programs could be syntactically based matching of small set of facts relevant for the ongoing common activity.

The acceptance of the speech act offer can be communicated in the following speech act. But in some situations the receiver program may criticize the claimed validity, or request additional information. The program that made the speech act offer is obliged to redeem its claims or provide the requested information. Allowing the process of reaching agreement to span over several speech act is conceptually simple extension, but it opens the bottom-up approach for structuring the threads of communication. When the programs with this feature are communicating about the composite entities, their threads of communication would exhibit hierarchically organized discourse structure [2].

5 Computing the response

We started with the simple reactive system. It is easy to write programs with such simple structure, but when more complex behavior is required and the number of rules is large, they are hard to control. With the introduction of speech acts we have clear separation between the illocutionary and perlocutionary effects. The introduction of communicative action provides for segmentation of programs into parts relevant for the desired computation and parts relevant for communication.

Simple reactive communicating programs accept

stated facts or obey the issued commands after checking the satisfaction of the small set of conditions. This programs can be implemented using the functional programming style only. Interaction with them can be seen as a kind of a remote function evaluation.

More complex approach in deciding on a response is to use one of pattern-based algorithms as we saw in the case of Elephant language. Another approach is to use a variant of logical programming. One such approach is taken by Kowalsky an Sabry [5]. Their aim is more ambitious than making just a communicative system. They want to extend logical programming so that it becomes a comprehensive foundation of computing. They argue that the lack of reactive component is the main deficiency of logical programming. As a remedy they propose the introduction of the agent cycle in logical programs. The agent cycle combines observations, that may trigger some action, with a logical programming style reasoning. The agent cycle has to be executed within the limited period of time. Communication in multi-agent systems is achieved by exchanging messages that agents can "observe".

If we use the ideas of communicative action to organize interaction and computation of Elephant programs, using logical reasoning in practical systems looks feasible. The problem of using automated reasoning is its difficulty of control. In communicative action there is a clear separation of the control elements and declarative part. The propositional component is a logical sentence whose truth have to be determined, or we have to find one or more of its instance (fill in answers). The propositional component is processed against the background context of discourse that is a small set of sentences. The illocutianary component is used for determining the parameters of automated reasoning. One possible inference procedure can be the resolution procedure with the suitable selected strategy. We use propose using automated reasoning only as a mechanical, entirely syntactic process.

6 Keeping the traces of events

The runtime system of the Elephant 2000 program keeps a virtual history list of past events[3]. McCarthy provides less details about this feature of the language. He calls the history list a virtual list because its lisp structure will be hidden from the language users. The language will provide ways of referring to the past that will be semantically more like the ways people refer to the past in natural languages than just having an index variable into a list [12].

On the first glance, recording the events of interactions and some results of computations looks like a simple feature of the language. However, if we allow that the runtime system of the Elephant program not only records particular events but also use this records in determining future responses, we are enabling much higher levels of complexity. It is hard to predict how will such historically "aware" systems behave. Can we control such systems? Does it mean we can do programming by "talking" to communicating systems? We can tell the Elephant program the sentence that can be used in the as the rule. The program decides whether to accept or not the offered sentence by checking the rules of acceptability. Or, the program may asks us some additional information [1]. It looks that Lisp with embedded Elephant brings us closer to the Advice Talker system[10] that ordinary Lisp.

When the assimilation of the new sentences in the data base of the system is allowed, it is necessary to maintain the efficiency of the system. Experiences with rule base system teaches us that that they work fine with small set of rule and facts but do not scale well on more complex problems. An approach to solving this problem could be to follow some hints from theories of communication. In principle, we can interpret the sentence we hear using the total set of assumptions we acquired through our life experience. But in the actual communicative situation we determine a particular limited context (horizon) of assumption that is relevant for understanding the message [3].

For the purpose of designing communicative system we can assume that the context is the set of rules and facts in case of rule-based systems like Elephant runtime system or a set of premises in logic-based systems. We can use some recent research results to find a practical solution to the problem of determining the appropriate context of communication and keeping the size of contex small. McCarthy proposed an approach to formalizing contexts [13]. Dan and Sperber [16] developed

[3] *An elephant never forgets!* (a quote from [12])

interesting theory of communication based on the concept of relevance. One part of their theory is concerned with explaining how is the context initially selected and how it changes in the ongoing process of communication. The Dan and Sperber's theory of communication uses syntactically-based approach to logic. To my knowledge they didn't try to make a computational model of their theory. It is an open question how efficient their method of controlling context can be.

Keeping the traces of events could be the most important ingredient in developing communicating systems that can engage in communication similar to the communication in sentences of natural languages. People not only engage in conversation to achieve something, but in the process of conversation necessarily change. For Maturana[8], the linguistic interaction triggers structural changes of the participants of the interaction that brings forth the phenomena we label as understanding, meaning or representation of the world. For Gadamer [3], any event of coming to agreement includes the accumulation of experiences and can cause changes of the set of background knowledge of the participants. Entities of the world can be conceptualized in an infinite number of ways. Humans are finite systems that must take a concrete conceptualization. To reach the agreement, interacting individuals, with conceptualizations that differ to some degree, have, in some cases, to change existing assumptions and/or assimilate new assumptions. Gadamer call this process "horizon fusion".

7 Proving correctness of Elephant programs

High level features of Elephant language do not preclude the mathematical simplicity of the language. One problem with the methods for proving correctness of computer programs is that they appear too complex to people without mathematical inclinations. If it is successfully developed, the elephant method may become more acceptable because it will be either automatic or its control can be hidden behind a sequence of simple questions. In comparison with other methods for computer proof-checking, the elephant method doesn't require a special theory of programming. Another benefit of it is that it is not necessary to annotate the statements of the program with additional statements.

There are at least two good reasons for having a good method to prove the correctness of interacting programs. First, allowing the changes of interacting programs in the process of communications implies the possibility of programming-by-conversation. Having an automatic method for checking partial correctness of the modified program improves the stability and "trustfulness" of the system. Second, even the simple static program that interact with other programs in behalf of individuals or companies must have the high level of correctness to be trusted.

McCarthy distinguishes between the input-output specifications of program correctness and accomplishments specifications. The accomplishments specifications specify what is expected of programs to achieve interacting with other programs or performing actions in the real world. Input-output specifications define what it means that program accept or fulfill commitment or answer a question truthfully. Input-output specifications are related to illocutionary acts and they are easier to verify than accomplishment specifications. Some sentences of input-output specifications can be generated automatically from the form of the program.

Elephant method of proving the correctness of programs is based on the facts that program sentences are logical sentences in which time is used explicitly. McCarthy demonstrates it in the simple setting without using the communicative features of the language. He considers the fragment of a simple ALGOL program that computes the product mn by initializing the partial product p to 0 and adding to it the value of m n times.

```
0  start :  p := 0;
1           i := n;
2  loop :   if i = 0 then go to done;
3           p := p + m;
4           i := i − m;
5           go to loop;
6  done :
```

We assume that the values of m and n are already initialized. One way to express this fragment of a

program is to express variables p and i as functions of time [12].

$$\begin{aligned} p(t+1) \quad = \quad & \texttt{if } pc(t) = 0 \texttt{ then } 0 \\ & \texttt{else if } pc(t) = 3 \texttt{ then } p(t) + m \\ & \texttt{else } p(t), \end{aligned}$$

$$\begin{aligned} i(t+1) \quad = \quad & \texttt{if } pc(t) = 1 \texttt{ then } n \\ & \texttt{else if } pc(t) = 4 \texttt{ then } i(t) - 1 \\ & \texttt{else } i(t), \end{aligned}$$

$$\begin{aligned} pc(t+1) \quad = \quad & \texttt{if } pc(t) = 2 \wedge i(t) = 0 \texttt{ then } 6 \\ & \texttt{else if } pc(t) = 5 \texttt{ then } 2 \\ & \texttt{else } pc(t) + 1. \end{aligned}$$

The function $pc(t)$ acts as the "program counter". Under the straightforward interpretation of the above statements as recursive functions the system can correctly compute the desired product mn. The correctness of the program can be determined proving the sentence

$$\begin{aligned} \forall m\ n\ (n \geq 0 \quad \supset \quad & \forall t(pc(t) = 0 \supset \\ & \exists t'(t' > t \quad \wedge \quad pc(t') = 6 \wedge p(t') = mn))). \end{aligned}$$

The previous sentence can be proved using the axioms of arithmetic and the axiom schema of mathematical induction.

8 Conclusions

I didn't provide a blueprint for constructing communicating system. I described their ingredients. With the insights from several theories of communication we can expect that the appropriate composition of this ingredients will make an useful communicating program. This is not a magical procedure for writing programs. It is a well known exploratory bottom-up style of programming in Lisp.

It appears that the programming language features necessary for developing AI programs are also the features useful for developing communicating programs. Logical programming languages like Prolog can be considered as the alternative to Lisp for programming communicating systems. A particularly strong competitor could be a multiprocessing (or concurrent) logical programming language with the flexible search procedure. Certainly, for any more complex patterns of communication we will have large parts of programs in the declarative form. However, at this stage of experience with communicating programs I see Lisp as more appropriate language. The relationship between the logic and control in communicating programs is not yet clear. Not having to write the search procedures from the scratch is less important than having the flexibility in developing the appropriate control and being able to efficiently implement the history list of the system.

It shouldn't be surprising that Lisp appears as specially designed for writing communicating programs. From the beginning Lisp was indirectly connected with communication. Lisp "was designed to facilitate experiments with a proposed system called the Advice Taker" [11]. The behavior of Advice Taker "will be improvable merely by making statements to it, telling it about its symbolic environment and what is wanted from it." [10]

References

[1] Zoran Aleksić. *Programming, communicative action, and understanding.* Research Report. NUCBA Bulitten. Vol. 41, No. 2, 181-193, 1999.

[2] James Allen. *Natural Language Understanding.* The Bemjamin/Cummings Pub. Com., 1995.

[3] Hans-George Gadamer. *Truth and Method.* The Continuum Pub. Comp., 1994.

[4] Jürgen Habermas. *On the Pragmatics of Communication.* The MIT Press, 1998.

[5] Robert Kowalski and Fariba Sadri. *From Logic Programming towards Multi-agent Systems,* To appear in: Annals of Mathematics and Artificial Intelligence, Academic Press, 1999. Available at http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html.

[6] Labrou Y., Finin T., and Peng Y.. *Agent Communication Languages: The Current Landscape* IEEE Intelligent systems, vol. 14. no. 2, 45-52, 1999.

[7] Jon Mallery. *Common Lisp Hypermedia Server,* 1994. Available at http://www.ai.mit.edu/projects/iiip/doc/cl-http/server.html

[8] Humberto Maturana. *Biology of Language.* In George Miller and Elizabeth Lenneberg, ed., Psychology and Biology of Language and Thought, Academic Press, 1978.

[9] George Herbert Mead. *Mind, Self, and Society.* The University of Chicago Press, 1934.

[10] John McCarthy. *Programs with Common Sense.* 1958. Available at http://www-formal.stanford.edu/jmc/.

[11] John McCarthy. *Recursive Functions of Symbolic Expressions and their Computation by Machine (Part I).* Commun. of the ACM, April 1960. Available at http://www-formal.stanford.edu/jmc/.

[12] John McCarthy. *Elephant 2000: A Programming Language Based on Speech Acts.* Available at http://www-formal.stanford.edu/jmc/.

[13] John McCarthy. *Formalizing Context (Expanded Notes).* Available at http://www-formal.stanford.edu/jmc/.

[14] John McCarthy. *Formalizing Common Sense: Papers by John McCarthy.* Ablex Publishing Corporation, 355 Chestnut Street, Norwood, NJ 07648, 1990.

[15] John R. Searle. *Speech Acts.* Cambridge University Press, 1969.

[16] Dan Sperber and Deirdre Wilson. *Relevance: Communication and Cognition.* Blackwell, 2nd ed., 1995.

[17] Lev S. Vygotsky. *Thinking and Speech.* Plenum Press, 1987.

[18] J. Weizenbaum. *ELIZA*. Commun. of the ACM,9:36-45 1966.