Research Report AI–1995–01

Toward a New Type of Language for Electronic Commerce

Michael A. Covington

Artificial Intelligence Center
The University of Georgia
Athens, Georgia 30602–7415 U.S.A

Toward a New Type of Language for Electronic Commerce

Michael A. Covington
Artificial Intelligence Center
The University of Georgia
Athens, Georgia 30602–7415 U.S.A.

Abstract

This paper surveys practical issues in the design of a formal language for business communication (FLBC) in which transactions are put together by combining meaningful elements, much as a programming language encodes algorithms. Such a language is preferable to existing codes such as ANSI X.12 and UN EDIFACT because of its much greater versatility. The new language is tentatively named LEC (Language for Electronic Commerce).

1 The problem

1.1 Why a new language?

Electronic data interchange (EDI) is the automated exchange between computers of the messages needed to carry out business transactions. Examples include electronic funds transfers, purchase orders, invoices, and various kinds of administrative information. EDI users range from corporations to armed forces to museums and libraries.

Present-day EDI messages look like Figure 1, and I want to make them look like Figure 2. The difference is more than cosmetic. Current EDI standards such as ANSI X.12 [1] and UN EDIFACT [2] are essentially *data formats*, consisting merely of data fields arranged on a predefined form. A new type of form is needed for each type of message — so that in practice, there are hundreds — and if a message doesn't fit a predefined form, you can't express it at all. For example, Moore [3] found it impossible to use EDI to tell a customer that, contrary to an earlier message, a shipment had not actually gone out.

In this paper I want to sketch a new Language for Electronic Commerce (tentatively named LEC) in which messages are built by combining meaningful elements.[1] Essentially, LEC will work like a programming language except that instead of describing algorithms, it will describe business transactions. Messages are not limited to predefined forms; instead, just as in programming languages, the elements of the language can be put together in any meaningful way.

1.2 From invention to design

Languages of the type that I envision have been prototyped by Moore [3], Kimbrough and Lee [4], Dewitz and Lee [5], and probably others. The development of such a language was apparently first advocated by John McCarthy [6].

In this paper I want to move from the problem of *inventing* this new type of language to the problem of *designing* a practical language of this type. In so doing, I will draw upon applicable knowledge from natural language semantics and pragmatics, programming language design, artificial intelligence, and other fields. I will not be presenting a complete design, merely an approach to a set of design problems. For brevity I will sometimes use the name LEC to encompass not only the language I hope to design, but also any other languages that share its relevant properties.[2]

LEC breaks with traditional business data processing technology in two ways. Like programming languages, LEC uses recursively defined syntax rather than flat data formats, thereby enabling a smaller language to do a bigger job. More importantly, LEC assumes that messages will be processed by inference, not just decoding [3,7]. Whereas conventional EDI merely copies a block of data from one computer into another, LEC encodes messages that the receiving computer must figure out how to handle. In this respect LEC is much more like human language. When I talk to you, I am not copying my thoughts into your mind; rather, I'm giving you a message that you can decide to handle in any number of ways, depending on the context, whether you trust me, and so forth. Likewise, LEC is designed for computers that will decide, in various sophisticated ways, how to handle incoming messages, rather than copying them blindly into internal databases.

The requisite technology for implementing inference, logic programming, is now twenty years old, and adequate

[1] I want to thank Steve Kimbrough, Roggie Boone, and three anonymous referees for assistance and encouragement with this project. I am solely responsible for errors that remain.

[2] The generic term, FLBC (formal language for business communication), is also the name of a language designed by Moore [3].

```
ST*840*159
BQT*00*Q47391*820430
N1*SE*X, Inc.
N1*BY*Y Co.
P01*1*30000*EA*0.42*PN*747355*PD*Circuit Network
SCH*10000*EQ****002*820604
SCH*20000*EA****002*820709
CCT*1*30000
SE*9*159
```

Figure 1: Example of an EDI transaction in ANSI X.12 format, requesting a price quote on 30,000 circuit networks with specified delivery dates [15].

```
lec(
   dialect(1.25,full),
   from="Y Co.",
   to="X, Inc.",
   content:
     we request:
       you inform us:
            price_of:
                 item_1 := (part_no:747355, description:"Circuit Network"),
                 action=
                   deliver(from=you, to=us, item=item_1, qty=20000, date≤82/06/04) &
                   deliver(from=you, to=us, item=item_1, qty=20000, date≤82/07/09)
)
```

Figure 2: What the new type of language may look like. Semantic and pragmatic analysis is needed before this language can actually be designed in workable form.

computer power is now available in machines as small as 4×6 inches (10×15 cm).[3] Accordingly, this aspect of LEC will be an application of known artificial intelligence techniques.

1.3 How communication can fail

The central role of inference implies that LEC will have a possible failure mode that conventional EDI does not. If your software is designed to handle a particular conventional EDI form, you can be confident that it will always succeed in doing so. The only reasons an incoming message might be un-handleable are that it violates the syntax of the form, or else the data within it is either erroneous or malformed. Valid messages without erroneous content will *always* be handled correctly.

But with a language that requires inference, it is possible to receive a perfectly well-formed message containing a perfectly reasonable request which you nonetheless cannot handle, not because you lack the ability to comply, but because *you can't figure out what to do with it* — the requisite inferences are not within your computational ability, or you lack some background knowledge that the sender assumed you had. This is a very familiar scenario in human-to-human communication. How it will affect EDI remains to be worked out. Obviously, one of the desiderata for an inferential message handler is the ability to troubleshoot such situations automatically and reply with appropriate requests for clarification.

Suspicions about this failure mode probably account for some of the skepticism that is occasionally expressed about new-style EDI languages. But on closer examination, this failure mode is not as pernicious as it sounds, because, in general, the messages that fail this way are messages that could never even have been attempted with conventional EDI. At worst, you are no worse off than with the old technology. Almost all of the time you are better off.

1.4 Design process

To design LEC we need not only an adequate theory, but also an empirical study of the communicative power

[3] A $4 \times 6 \times \frac{1}{2}$-inch 80486 PC was exhibited at the 1995 Embedded Systems Conference in Atlanta. This is more than enough computer power for the purpose.

needed for electronic commerce. Here we can draw on recent developments in theoretical linguistics, especially formal semantics and formal pragmatics. Linguists now use set-theoretic techniques to specify, with mathematical precision, the relations between utterances, their meanings, and the situations in which they are used. Naturally, not all of human language has been analyzed this way, but we don't need all of it; we only need the subset that is essential for commercial transactions.

Some important work along these lines has already been done [3,4,5,8,9,10]. In the remainder of this paper I shall sketch some applicable concepts from linguistics and computer science and make concrete proposals for the design of a new language.

2 Language and Communication

Figure 3 shows a model of how the parts of any language fit together:

- Semantics relates the message to the things it can talk about (its logic and ontology);
- Pragmatics relates the message to the communicative situation;
- Syntax specifies how the parts of the message fit together.

This model is an adaptation of the standard five-level model of human language used by linguists [11,12], with physical encoding and transport taking the place of morphology and phonology. The three-way distinction between syntax, semantics, and pragmatics goes back to a 1938 monograph by C. W. Morris [13], but the pragmatics of human language, in particular, is a new field, having developed in the last 30 years or less.

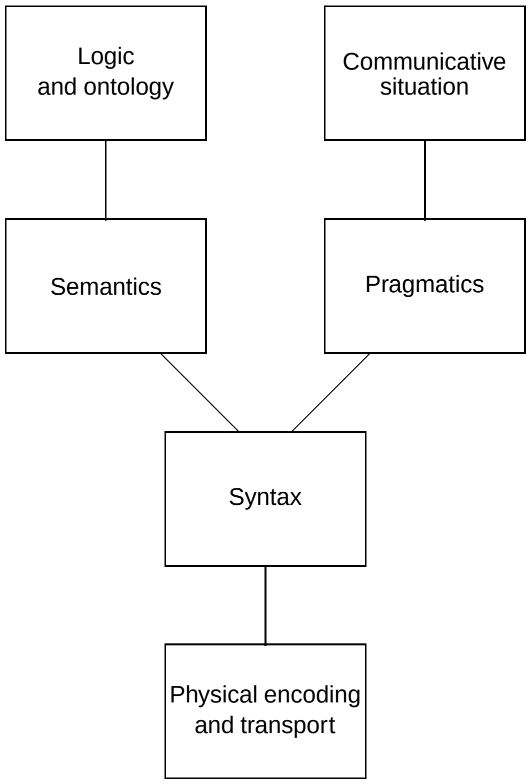

Figure 3: A model of language and communication.

3 Logic and ontology

Ontology is what you can talk about or think about, and logic is how you describe and reason about the properties and relationships of things. The design of LEC must reflect the ontology and logic of business transactions.

3.1 Entities and relationships

A striking characteristic of X.12 and EDIFACT is their bloated ontology. When the same entity or type of entity turns up in more than one place, the sameness is not recognized. To take an extreme case, EDIFACT has no concept of "number" — instead, there are 3–digit numeric fields in some places, 4-digit numeric fields in others, 10-digit numbers somewhere else, and so on. The problem, of course, is that EDIFACT does not distinguish concepts from their physical representations. In essence, EDIFACT is a language for depositing character strings into particular places on a remote computer, rather than a language for exchanging knowledge. X.12 is largely the same.

The ontology of LEC will be much leaner, but it would be premature to try to enumerate everything it will contain. Minimally, LEC will have to refer to numbers; quantities expressed in specific units (such as length, weight, currency, etc.); individual participants in a transaction; mechandise; and relations such as ownership, possession, and transportation. It is likely that the basic concepts of business transactions will be arranged in an 'is a' hierarchy with default inheritance of properties.

3.2 Facts and presumed facts

One obvious difference between practical knowledge and pure classical logic is that practical knowledge can be overruled. You can get information that says X and then, later on, get better information that says not-X. When this happens, you don't have a contradiction; you've merely expanded your knowledge.

This is known as defeasible or non-monotonic reasoning. Defeasible inference by computer is now a well established technology [14] and will be incorporated into the LEC inference engine. Defeasible reasoning in business communications has been explored in depth by Kimbrough and Moore [15] among others.

3.3 Extensions to classical logic

LEC and the computer systems that process it also need three familiar extensions to classical logic. First, in business you often have to talk and reason about what is *possible,* not merely what is the case today. That is known as MODAL LOGIC.

Second, you have to reason about time, and facts that are true at different times, using TEMPORAL LOGIC.

Third, you have to reason about obligations (both your own and other people's), using DEONTIC LOGIC. The application of deontic logic to commerce is already an active research area [15,16]. Kimbrough and Moore [15] point out that in real life, all obligations are defeasible; no matter what your obligations seem to be, there is always the possibility of finding out that they are really something else.

Again, computer inference techniques for these extensions to classical logic are available [17,18,19,20,21]. Full implementations of modal, temporal, and deontic logic will not be needed because LEC is not aiming for the full expressive power of human language, only a minimum level sufficient for business transactions. Accordingly, external limitations can be imposed to make automated reasoning more practical. Since so much research is being done on these topics by others, I shall not pursue them here.

4 Semantics and vocabulary

The semantics of LEC will be based on first-order logic, with the extensions already mentioned.

A major design issue will be the *atomization of meaning* [22] — that is, whether to break meanings down into smaller units, and if so, how. For example, to sell is to trade for money; to trade is to make a pair of transfers of possession in opposite directions each cancelling the debt created by the other; and so on. The practical question is which concepts should be treated as atomic, and which should be broken down further.

This is a recurrent issue in the design of programming languages; more generally it might be called SYSTEMATICITY VERSUS CONVENIENCE or CHOOSING ELEMENTS OF THE RIGHT SIZE. Consider for example the percent key on your pocket calculator. Taking a percentage is not an elementary mathematical operation, but it comes up so often in business arithmetic that it needs a button of its own. A calculator designed for theoretical elegance would not have a percent key (nor, perhaps a square root key) and would not be as handy in practical work.

Similarly, the vocabulary of LEC needs to provide for concise expression of the concepts that come up regularly in commerce, whether or not they are equivalent to combinations of things already provided for. On the other hand, the language should be systematic enough that programmers can remember how to use it and computer implementations are reasonably clean and simple. The tension between theoretical elegance and practical usefuless will always be felt.

5 Pragmatics

Pragmatics is the relation between a message and the situation in which it is uttered. In LEC, pragmatics comprises speech acts, conversational maxims, and a number of more mundane aspects of message handling.

5.1 Speech acts

Austin [23], Searle [24], and others have identified various ILLOCUTIONARY ACTS that one can perform by uttering a message. On one scheme, illocutions in natural langauge fall into five major types, assertions, promises, instructions, declarations (e.g., christening a ship, defining a new term), and expressions of feeling.

Not all of these occur in commerce. Further, some relatively specialized illocutions, such as offers, are so common that they should probably be treated as basic types. A basic set for electronic commerce might comprise the following:

- Informing (giving information)
- Confirming information already given
- Inquiring (requesting information)
- Requesting or commanding action
- Promising (obligating oneself)
- Making an offer
- Accepting an offer
- Defining a new term

The first seven of these were prominent in Moore's empirical study of EDI speech acts [10]. The last one is a hook for extending the language. On the application of speech act theory to commerce, see also [25].

5.2 Conversational maxims

In a seminal paper, Grice [26] pointed out that in ordinary conversation, people abide by a number of maxims such as "Be relevant," "Be concise," and "Be truthful (not misleading)." To deduce what an utterance means, we routinely presume that these maxims are being followed.

Grice's maxims may be the key to an important part of the inferential process for handling LEC messages. Fortunately, commerce is much less subtle than natural language in general, and in commerce there is a strong tendency to make messages explicit. One does not have to deal with insinuations, rhetorical understatements, metonymy, or other figures of speech. The prominent inferential tasks have rather to do with associating the message with its sender and the relevant background knowledge. The overriding conversational maxim in commerce may well be something like, "Treat this message like other messages of the same type unless there is a demonstrable reason not to do so."

At least I hope it will remain that simple. Kimbrough (personal communication) has pointed out the possibility of an artifically intelligent EDI system "learning" to exaggerate or lie in order to get results. The system on the other end would then learn to discount its assertions. The process could escalate until there was a complete failure of communication. (We see this happen between humans.) We would want our inferential systems to detect such situations and head them off somehow.

6 Syntax

6.1 Does syntax matter?

One could argue that the syntax of LEC is almost a moot point; any ad hoc representation that has the right semantics and pragmatics will do. After all, LEC is for computer-to-computer communication, and its readability or elegance as judged by human eyes is unimportant.

That, however, is not true. Human beings have to implement the language even if they will not be the ultimate senders or recipients of messages. The history of programming languages shows that seemingly petty decisions about syntax and physical encoding can have a large effect on the success or failure of a language [27]. For example, the designers of ALGOL 60 failed to specify the details of physical encoding, and as a result, ALGOL programs were never portable [28]. Fortran and COBOL, with their well-specified physical representations, carried the day even though ALGOL was a more sophisticated language. Again, one of the factors that confined PL/I to IBM hardware was its reliance on the EBCDIC character set. And the main reason most of us have never used APL is its insistence on a character set all its own.

On the other hand, the success of PostScript [29] as a computer-to-computer language is attributable to many good design decisions, among them free-form layout, the identification of the language in the first four characters of every program, and the ability to mix MS-DOS and UNIX end-of-line marks.

6.2 Prolog base

Following Moore [3] and others, I adopt Prolog as the basis for LEC. Prolog is the only major computer language that is designed to represent inferential knowledge as well as algorithms. Unlike conventional languages, Prolog has a written representation for every value of every data type.[4]

The syntax of LEC will be a subset of that of ISO Prolog [30]. Prolog, in turn, is based on C. Thus, familiar tokenization and parsing algorithms can be used. Because Prolog can be used as its own metalanguage, implementation of LEC in Prolog will be especially easy.[5]

For ease of language identification, each LEC message will be required to begin with its principal functor (probably `lec`) with no comments or white space preceding.

LEC messages will be text, written in the 128-character ASCII character set. Conversion to or from other character sets is left to the transport protocol.

Why not use a more concise binary encoding rather than text? For two reasons. Text is readable and transportable; binary codes may not be. Second, compression of data into concise binary form can be carried out by the transport protocol. In any case, industry experience with another machine-to-machine language — PostScript graphics — has shown that massive prolixity is tolerable. PostScript files often start with many kilobytes of auxiliary definitions that are never used. Whether LEC will be subjected to comparable abuse remains to be seen.

Like Prolog, LEC will use free layout, and comments will be permitted. Unlike PostScript, LEC will not fall into the trap of redefining some comments to be significant elements as the language evolves. Instead, an open-ended message element (perhaps called `tag`) will be provided for implementing unforeseen extensions.

A standard printed representation for LEC messages will be devised so that a message transmitted with minimal or

[4]With a few arcane exceptions, of course, such as structures that are instantiated to structures that contain them.

[5]More than one person has suggested that I use SGML or one of its derivatives, such as HTML, instead of Prolog. This would not be advantageous. SGML is a language for describing printed text, not exchanging knowledge, and it does not have the right syntactic units to encode the ontology of LEC. (Instead of numbers, structures, and lists, everything is just character strings.) Moreover, SGML is verbose; instead of `msg(abc)` one would have to write `<MSG> abc </MSG>`. It would of course be trivial to translate Prolog-based LEC into a semantically equivalent notational variant based on SGML.

nonstandard use of whitespace can be printed automatically in readable form.

6.3 Some fine points

Prolog distinguishes between atomic symbols and character strings. Accordingly, LEC will use Prolog atoms for meaningful words, and character strings for free text and for names, addresses, and similar material that is merely reproduced without being interpreted. Because double quoted strings are byte strings, they can contain any type of data.

Many of the predicates used in commerce have multiple arguments, some of which are optional or unknown. For example, the act of selling might include a seller, a buyer, a thing sold, a price, and a date and time, but it is common to leave some of these unspecified. Instead of standard Prolog notations such as

```
sell(you,us,"Circuit Network",
             (85,usdollars),_,_)
```

LEC will use structures with labeled arguments, such as

```
sell(from=you,
     to=us,
     item="Circuit network",
     price=(87.5,usdollars),
     date=...
    )
```

or even discontinuous predicates as in:

```
you sell us "Circuit Network"
for (87.5,usdollars)
```

Here `A sell B C for D` is a template that matches the predicate-argument structure.

To eliminate the unreadable pileup of parentheses that occurs at the end of a formula such as

```
request(we,inform(you,us,whether(
     possible(sell(you,us,...))))))
```

LEC will modify the Prolog operator table to use colons in a special way. A colon will indicate that everything following it is a single argument, until the end of the expression or a higher-level closing parenthesis is found. Thus, instead of `a(b(c(d)))` one can write `(a:b:c:d)`.

Using these enhancements, the expression at the beginning of this section can be written

```
we request:
  you inform us whether:
    possible:
      you sell us ...
```

greatly improving readability.

7 Transport and Message Management

By "transport" I mean the mechanism outside the language that is responsible for getting messages from place to place. Transport includes communication protocols (SMTP, FTP, etc.), character set conversion, reformatting to comply with line length limits, and the like.

There is some duplication of function between the transport layer and the pragmatic information encoded in the message itself. For example, the sender and the addressee are identified in both places. But this duplication of function is not undesirable. First, deliberate discrepancies may be useful. For example, a message may need to be addressed (internally) to any of several different processes or functions at the same (transport-layer) email address. Second, by checking for unexplainable discrepancies, one can detect errors. The failures of the transport layer will not compromise the contents of a message.

Transport need not be trivial and mechanical. Moore and Kimbrough [31] point out the advantages of using a MESSAGE MANAGEMENT SYSTEM to automate message-handling tasks that would otherwise require human intervention. LEC is an ideal language for such a system because each LEC message can be understood, partly or by completely, by each message management system that it passes through.

8 What next?

8.1 The LEC project

To actually design and implement LEC it will be necessary to work through a corpus of actual business transactions, identifying the communicative power needed and designing a language equal to the task. Fortunately, much of the data-gathering has been done by the designers of existing EDI standards. EDIFACT and X.12 are, after all, nothing but corpora of transaction types. The real work now is to analyze the semantics and pragmatics of the EDIFACT and X.12 transactions, just as the inventors of Fortran and COBOL studied existing assembly language computer programs.[6]

8.2 Applications of LEC

The most obvious use for LEC is, of course, as a more versatile substitute for X.12, EDIFACT, and similar data languages. As such, it has the same advantages, including the ability to automate routine commerce and the ability to do business without having to know English or another

[6]The University of Georgia Research Foundation is seeking funding for this work.

major world language (it's no accident that EDIFACT is popular in Hungary and Bulgaria).

Because of its versatility, LEC also lends itself to automatic translation into and out of human languages. That is, it will be quite feasible to generate English, French, or Hungarian translations of LEC messages. Further, because LEC is logic-based, it will be practical to go the other way, writing software that accepts human-language input (about routine commercial matters, of course, not full unrestricted English) and converts it into LEC messages. Like human language, and unlike X.12 or EDIFACT, LEC messages are constructed by putting meaningful elements together. Thus, one does not have to interpret the entire natural language input and then go searching for a form that fits it; rather, one has only to map the meaningful elements of one language onto those of another.

8.3 Criteria of success

LEC has two design goals: to substitute for traditional EDI formats, and to deliver functionality that traditional EDI cannot. Testing of prototypes will focus on these goals. Types of software that I envision building in order to test LEC include:

- Translators to convert a subset of X.12 into LEC;
- Translators to convert LEC into X.12, as far as possible (a much harder problem because LEC can say things that X.12 cannot);
- Automatic message handlers and transaction processors to accept and respond to LEC messages;
- Translators to translate LEC messages into (approximations of) English and other human languages.

Eventually, of course, LEC should be tested in a real or simulated business situation analogous to the bicycle shop of Kimbrough and Moore [3].

References

[1] *Electronic data interchange: X12 standards, draft version 3 release 4.* New York: American National Standards Institute, 1993.

[2] Berge, John. *The EDIFACT standards.* Manchester (England): NCC Blackwell.

[3] Moore, Scott A. *Saying and doing: uses of a formal language in the conduct of business.* Dissertation, Ph.D., University of Pennsylvania, 1993.

[4] Kimbrough, Steven O., and Lee, Ronald M. On illocutionary logic as a telecommunications language. Proceedings, Seventh International Conference on Information Systems (1986), 15–26.

[5] Dewitz, Sandra K., and Lee, Ronald M. Legal procedures as formal conversations: contracting on a performative network. Proceedings, Tenth International Conference on Information Systems (1989), 53–65.

[6] McCarthy, John. The common business communication language. Albert Endres and Jürgen Reetz, eds., *Textverarbeitung und Bürosysteme.* Munich: Oldenbourg, 1982.

[7] Bach, Kent, and Harnish, Robert M. *Linguistic communication and speech acts.* Cambridge, Mass.: MIT Press, 1979.

[8] Kimbrough, Steven O., and Moore, Scott A. On automated message processing in electronic commerce: speech act theory and expressive power. Ms., University of Pennsylvania, 1994.

[9] Moore, Scott A. A communication framework for applications. Proceedings, Hawaii International Conference on System Sciences (1995).

[10] Moore, Scott A. Testing speech act theory and its applicability to EDI and other computer-processable messages. Working paper, U. of Michigan, 1995.

[11] Allen, James. *Natural language understanding.* Redwood City, Calif.: Benjamin/Cummings, 1995.

[12] Covington, Michael A. *Natural language processing for Prolog programmers.* Englewood Cliffs, N.J.: Prentice-Hall, 1994.

[13] Morris, C. W. *Foundations of the theory of signs.* (International encyclopedia of unified science, vol. 1, no. 2.) Chicago: University of Chicago Press, 1938.

[14] Ginsburg, Matthew L., ed. *Readings in nonmonotonic reasoning.* Los Altos, Calif.: Kaufmann, 1987.

[15] Kimbrough, Steven O., and Moore, Scott A. On obligation, time, and defeasibility in systems for electronic commerce. Proceedings, Hawaii International Conference on System Sciences (1993), vol. 3, 493–502.

[16] Lee, Ronald M. Bureaucracies as deontic systems. *ACM Transactions on Office Information Systems* 6 (1988) 87–108.

[17] Gabbay, Dov M.; Hogger, C. J.; and Robinson, J. A., eds. *Handbook of logic in artificial intelligence and logic programming.* 3 vols. Oxford: Oxford University Press, 1993-94.

[18] Lewis, Lundy Michael *The ontology, syntax, and computability of deontic logic.* Dissertation, Ph.D., University of Georgia, 1986.

[19] Meyer, J. J., and Wieringa, R. J., eds. *Deontic logic in computer science.* New York: Wiley, 1993.

[20] Raskin, J. F.; Tan, Y. H.; and van der Torre, L. W. N. *Modeling deontic states in Petri nets.* Report WP–1994–12–01, EURIDIS, Erasmus University, Rotterdam.

[21] Bons, R. W. H.; Lee, R. M.; Wagenaar, R. W.; and Wrigley, C. D. *Modeling interorganizational trade procedures using documentary Petri nets.* Report RP–1994–10–01, EURIDIS, Erasmus University, Rotterdam.

[22] Bolinger, Dwight. The atomization of meaning. *Language* 45 (1965) 555–573.

[23] Austin, J. L. *How to do things with words.* Oxford: Clarendon Press, 1962.

[24] Searle, J. R. *Speech acts.* Cambridge: Cambridge University Press, 1969.

[25] Auramäki, Esa; Lehtinen, Erkki; and Lyytinen, Lalle. A speech-act-based office modeling approach. *ACM Transactions on Office Information Systems* 6 (1988) 126–152.

[26] Grice, H. Paul. Logic and conversation. Peter Cole and Jerry L. Morgan, eds., *Syntax and semantics*, vol. 3: *Speech acts*, 41–58. New York: Academic Press, 1975.

[27] Wexelblat, Richard L., ed. *History of programming languages.* New York: Academic Press, 1981.

[28] Bemer, R. W. A politico-social history of Algol. Mark I. Halpern et al., eds., *Annual review in automatic programming* 5, 151–237. Oxford: Pergamon, 1969.

[29] Reid, Glenn C. *PostScript language program design.* Reading, Mass.: Addison-Wesley, 1988.

[30] Scowen, Roger, ed. *Prolog — part 1, general core.* ISO/IEC 13211–1:1995. Geneva: International Organization for Standardization.

[31] Moore, Scott A., and Kimbrough, Steven O. Message management systems at work: prototypes for business communication. *Journal of Organizational Computing* 5.2 (1995) 83–100.