PUC

Serie: Monografias em Ciência da Computação

Nº 03/77

(antiga/formerly: Monographs in Computer Science
and Computer Applications)

ON THE USE OF A PROBLEM STATEMENT LANGUAGE IN A SOFTWARE DEVELOPMENT ENVIRONMENT

by

Carlos J. Lucena
Romeu Delaroli
Vilmondes Gomes da Silva

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Marquês de São Vicente 225 — ZC 19

Rio de Janeiro — Brasil

Série: Monografias em Ciência da Computação

Nº 03/77

(antiga/formerly: Monographs in Computer Science and Computer Applications)

ON THE USE OF A PROBLEM STATEMENT LANGUAGE IN A SOFTWARE DEVELOPMENT ENVIRONMENT*

by

Carlos J. Lucena
Romeu Delaroli
Vilmondes Gomes da Silva

Series Editor: Michael F. Challis

April, 1977

* This work was supported in part by the Brazilian Government Agency FINEP.

For copies contact:

Rosane T.L. Castilho
Head, Setor de Documentação e Informação
Depto. de Informática - PUC/RJ
Rua Marques de São Vicente, 209 - Gávea
20.000 - Rio de Janeiro - RJ - Brasil

INFORMATION TO OUR READERS

We have decided to change the title of our series "Monographs in Computer Science and Computer Applications" to "Monografias em Ciência da Computação", to take effect from the first issue of 1977.

The aim and scope have been maintained, but besides including works in English we will also include works in Portuguese. As we are aware that problems concerning language barrier can affect our purpose of communicating our findings to our foreign audience, whenever we publish a work in Portuguese an abstract in English will be included.

Thank you for your interest concerning our publications during the last nine years.

DEPARTAMENTO DE INFORMÁTICA
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

INFORMAÇÕES AOS NOSSOS LEITORES

A nossa série "Monographs in Computer Science and Computer Applications" teve o seu título mudado para " Monografias em Ciência da Computação ", a partir do primeiro fascículo de 1977.

A série de " Relatórios Técnicos " foi absorvida pela série de "Monografias...", tendo sido o seu último fascículo, RT-01-77, publicado em janeiro deste ano. Será, portanto, através da série "Monografias em Ciência da Computação" que o Departamento publicará, em inglês ou português , todos os seus trabalhos de pesquisa de interesse à comunidade técnico-científica.

Agradecemos a todos pelo interesse demonstrado por nossas publicações durante os últimos nove anos.

DEPARTAMENTO DE INFORMÁTICA
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

ABSTRACT:

This paper describes a methodology which enables an effective use of the PSL/PSA system in a software development environment. A preliminary problem statement notation is interposed between the user (system's programmer) and PSL. A PSL problem statement can be systematically derived from this notation. The work also addresses the issue of making a specialized use of PSL to guide the production of a reliable implementation from the problem specification.

KEY WORDS:

PSL/PSA; software development; system's documentation; problem statement.

RESUMO:

Este trabalho descreve uma metodologia que possibilita uma utilização efetiva do sistema PSL/PSA, quando o mesmo é usado em um ambiente de desenvolvimento de software. Interpõe-se uma notação preliminar entre o usuário (programador de sistemas) e o PSL. Uma definição de problema em PSL pode ser derivada sistematicamente a partir desta notação. O trabalho também propõe um uso especializado do PSL que permite guiar a produção de uma implementação confiável, construida a partir da especificação.

PALAVRAS CHAVE:

PSL/PSA, desenvolvimento de software, documentação de sistemas, definição do problema.

CONTENTS

1. INTRODUCTION

The pair of languages PSL/PSA were originally designed as part of the ISDOS project at the University of Michigan. Their design objectives were respectively to formally express the specification of an Information System and to analyse automatically the specification documentation. The broad design goals of the Problem Statement Language (PSL) suggest a number of specializations of its use when particular Information Systems are taken into consideration. In particular, experiments conducted with PSL in the specification of software systems have suggested a number of methodological considerations for its effective utilization in this context. In this paper we propose a semi-formal and non-procedural documentation technique designed to precede a PSL specification. The documentation technique is analysable through procedures which are similar to the ones adopted by the PSA system and incorporates concepts which are familiar to software designers. A PSL problem statement can be systematically derived from the documentation technique. The authors conjecture that the proposed methodology contributes to the human engineering of the use of PSL for software development. Another dimension explored in this work is the systematic generation of COBOL programs from a PSL specification. The approach taken is that the semantics of PSL can be informally described in terms of schemas of COBOL programs used to derive an implementation from a PSL specification. The identification of the implementation rules for COBOL also highlights some restrictions that should be imposed on PSL programs when they are used for software design, and indicates an approach to the validation of programs constructed through the methodology from a PSL specification.

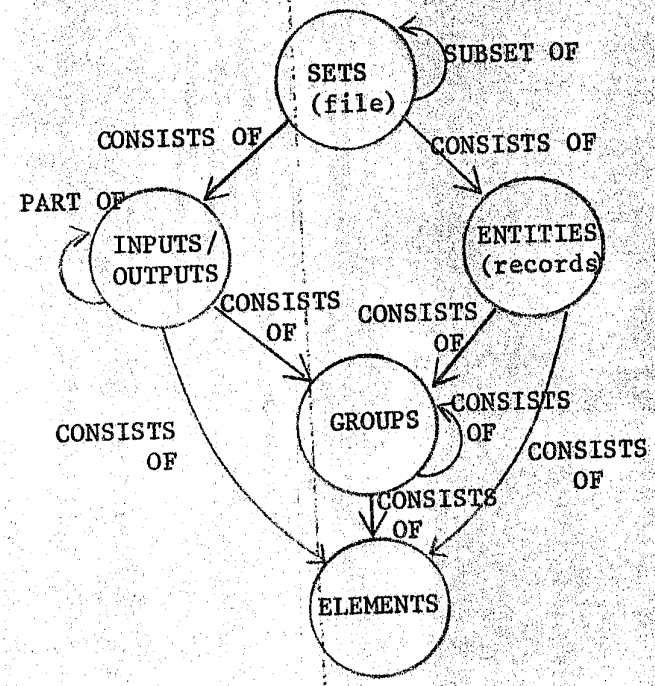
2. THE PROBLEM STATEMENT LANGUAGE

In this work we consider the PSL language [1,2] strictly as a specification notation for the expression of the system's programming level as it appears in application's programming. PSL at its present stage of development is mostly a powerful

documentation technique that allows for a more effective communication between system's analysts and their programmers. PSL has a formal syntax, a non-procedural structure and is designed in a way to support top-down design specifications. The semantics of PSL is based on a small number of concepts. These concepts are: the interfaces (produce inputs and receive outputs), the historical data definition facilities, the data definition facilities, the processes, the processes'control mechanisms and the system's parameters (such as size, time, volatility etc). A problem statement (or an application's software specification, in the present context) consists of various types of sections each specialized in the description of one of the above concepts.

It is possible to summarize the PSL semantics by saying that the "PSL abstract machine" has the following characteristics: both its processing and storage mechanisms have a network structure and a PSL problem statement is a textual description of the pair of networks which define the control flow and data structures which characterize a given data processing problem. The data structures which constitute the PSL machine storage are structured and related as in Fig. 1.

(a) Data Structure



(b) Data Relationships

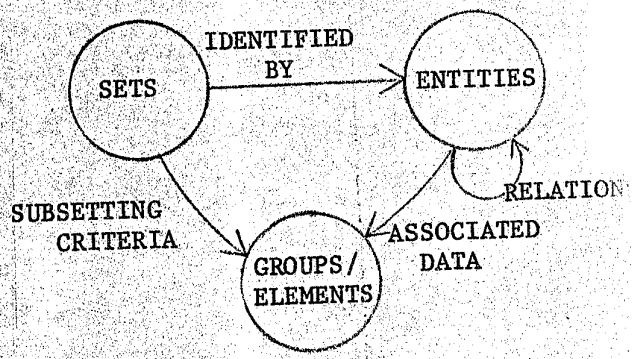


Fig 1 Examples of Data Structure and Data Relationships as Implied by a PSL Problem Statement.

The textual form of all the sections that deal with storage structures are presented schematically in Fig. 2.

INTERFACE SECTION

INTERFACE name
GENERATES input
RECEIVES output

INPUT (OUTPUT) SECTION

INPUT (OUTPUT) SECTION
GENERATED (RECEIVED) BY interface
USED BY process TO DERIVE data
DERIVED BY process USING data

SET SECTION

SET name
DERIVED (UPDATED) BY process USING data
USED BY process TO DERIVE (UPDATE) data
CARDINALITY IS system parameter

Fig. 2 Schemata of the PSL's Data Specification Sections

Processes are also expressed through specialized sections (Fig. 3) and the dynamic behavior of systems is modelled in a textual form that captures the ideas expressed through the graphic example in Fig. 4.

PROCESS SECTION

PROCESS name
RECEIVES inputs
GENERATES outputs
USES data TO DERIVE (UPDATE) data
MAINTAINS relation and/or subsetting criteria

Fig. 3 Schema describing a PSL process section

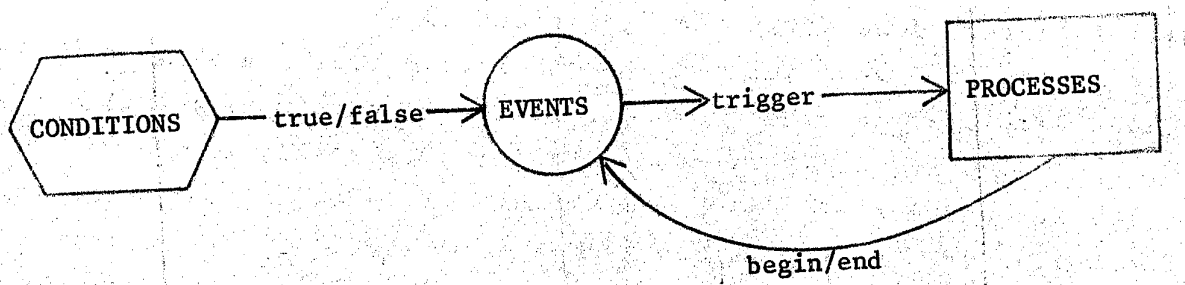


Fig. 4 Dynamic behavior implied by PSL's Control Mechanisms

System -parameters are used throughout a problem statement to express specification requirements related to time and space considerations as well as to considerations related to the volatility of certain historial information pertaining to the system.

3. FROM THE USER'S INTENT TO THE PROBLEM STATEMENT

The origin of the methodologies to be summarized in the sequel (they are completely described elsewhere [3, 4]) were several experiments that used PSL as a tool in the process of specification and implementation of several data processing problems. Most of the problems we observed during our experiments with PSL were related to the following major aspects:

a) PSL design assumes that a textual documentation can be used even at the earlier phases of programming systems' design (in fact, this assumption is stronger since PSL is addressed to general information processing systems). The difficulty is that the very nature of the language constructs that define the language impose an overhead to the software designer who seems to be inclined to resort to a tabular description of his design before start coding in PSL.

b) The PSA system checks in a very detailed form several aspects of the consistency and completeness of the specification. This

is, of course, done after the specification is complete. We contend that the verification process should take place as the design develops. In other words, a procedure should be available through which the designer could check incrementally each step in the formulation of his problem statement. By the very nature of the idea this cannot be performed automatically but, instead, should be (like structured programming) a methodology to be learned by the system's programmer. If this objective can be achieved, the designer would be better prepared to interpret the detailed PSA reports. The described procedure is analogous, at the program level, to informal program verification and the PSA reports would paralel the testing procedure that should normally follow it.

c) The large number of features comprised by PSL are justifiable only if we agree with its goals of "automatic" code generation and modelling of very general information processing systems. For the specification of a wide class of data processing problems, whose implementations are going to be encoded manually in a high level language, a number of simplifications are desirable in PSL so that it becomes a language in which a system's programmer may become "fluent".

As a solution to the above problems we have developed a methodology [3] which utilizes a set of standard forms to systematically gather the specification information which is necessary for the production of a PSL problem statement. This set of forms plays a role which is similar to the one played by the ADS system [5]which has been used in association with the SODA methodology [6]. The set of forms were designed having in mind the possibility of their systematic use for the generation (based on a set of rules) of a PSL "program". Likewise, a set of rules were proposed for the manual analysis of this preliminary specification . The central issues analysed by PSA can be preliminarly examined during the application of this step of the methodology. We did not want to automate this phase for the reason described in item c.

In Fig. 5 we display, in a condensed form, the set of forms used for the preliminary program specification.

1 GDF - FORM	REF	GENERAL DESCRIPTION FORM	AUTHOR SPEC.-	ALTER.-
SECTION:		SYSTEM:	DATE: VERSION	PG SEC NUM

2 IOD - FORM	REF	INPUT/OUTPUT DESCRIPTION	AUTHOR SPEC.-	ALTER.-
SECTION:		SYSTEM:	DATE: VERSION	PG SEC NUM

NAME	DESCRIPTION	REF. TO OTHER INFO

	ORIGIN					DISTRIBUTION		# PAGES	FREQUENCY
INPUT	INTERFACE	OCCURRENCE	VOLUME	FREQUENCY	OUTPUT	COPIES	INTERFACE		

RECORD	# OF RECORDS IN FILE	KEY	ATTRIBUTES		VOLATILITY		
			SIZE	RESTRICTIONS	ADDITION	DELECTION	UPDATE

CONSISTS OF THE FOLLOWING DATA

REF	LEVEL	NAME	REF	LEVEL	NAME	REF	LEVEL	NAME

3 FS - FORM		FILES SUMMARY	AUTHOR SPEC.-	ALTER.-
SECTION:		SYSTEM:	DATE: VERSION	PG SEC NUM

REF	NAME	DESCRIPTION	REF. OTHER INFO.	ATTRIBUTES			VOLATILITY	RESPONSIBLE FOR INTERFACE	RECORD NAME
				# REC	% GROWTH	SECURITY			

4 DD - FORM	REF	DATA DICTIONARY	AUTHOR SPEC.-	ALTER.-
SECTION:		SYSTEM:	DATE: VERSION	PG SEC NUM

NUMBER	NAME	DESCRIPTION	VALUES	TYPE CHECKING CRITERIA	FORMATS		
					I	ST	O

5 IPO - FORM	REF	INPUT PROCESS - OUTPUT FORM	AUTHOR SPEC.-	ALTER.-
SECTION:		SYSTEM:	DATE: VERSION	PG SEC NUM

INPUT	PROCESS	OUTPUT

6 SOD - FORM	REF	SYSTEMS DYNAMICS DESCRIPTION FORM	AUTHOR SPEC.-	ALTER.-
SECTION:		SYSTEM:	DATE: VERSION	PG SEC NUM

CONDITION NAME	DESCRIPTION	SOFTWARE COMPONENT	CONDITION FOR PROCESS INITIALIZATION

Fig. 5 Set of Forms for the Preliminary Problem Statement

Note that the format used in the IPO-form is inspired in the HIPO [11] methodology. The same top-down design approach supported by PSL is preserved in this earlier specification level. We envisage a procedure through which a system designer would carry his preliminary problem statement to a terminal and would interactively produce his PSL problem statement according to our proposed conversion rules. We have some preliminary evidence that this procedure contributes to the human engineering aspects of the PSL/PSA system.

We shall illustrate in the sequel one of the rules for producing a PSL text from the set of forms.

The strategy adopted in this step is to relate an EVENT to each CONDITION (either simple or composite) that enables a process. Using the SDD - form, processes and related conditions can be expressed as in Fig. 6.

SDD - FORM	——	——
——	——	——

Condition Name	Condition's Description
C1	A —— A
C2	B ——— B
C3	C ——— C

Process	Condition	Frequency
P1	C1.OR.C2	
P2	C3	

Fig. 6 Use of the SDD-Form

The derived PSL statement after the application of the conversion rule, will have the aspect displayed in Fig. 7.

```
CONDITION C1-OR-C2;
   TRUE WHILE
      A ——— A  OR  B ——— B;
EVENT BEGIN-P1;
   WHEN C1-OR-C2 BECOMES TRUE
   TRIGGERS P1;
CONDITION C3;
   TRUE WHILE
      C ——— C
EVENT BEGIN-P2;
   WHEN C3 BECOMES TRUE
   TRIGGERS P2;
```

Fig. 7 PSL statement derived from the SSD-Form.

In figure 8 we relate the several system's components and respective PSL sections to the various forms that constitute the preliminary problem statement

SYSTEM'S COMPONENT	RELATED PSL SECTION	FORM USED FOR THE PRELIMINARY PROBLEM STATEMENT
Interface	INTERFACE	General Description Form (GDF - Form 1)
Documents	INPUT	GDF and Input Description(Form 2)
	OUTPUT	GDF Output Description (Form 2)
Files	FILE	GDF and Files Summary (Form 3)
	RECORD	Record Description (Form 3)
Data Definition	GROUPS and ELEMENTS	Data Dictionary (Form 4)
Process Definition	PROCESS	GDF and Input-Process-Output Form (IPF-Form 5)
Process Control	Systems Dynamics	Systems Dynamics Description Form (Form 6)

Fig. 8 Relation between PSL sections and the set of forms used for the preliminary problem statement.

4. FROM SPECIFICATION TO IMPLEMENTATION

In attempting to gain in generality by moving away from current programming language concepts, PSL has adopted a semantic model which is very unfamiliar to most current application programmers. Still, one hopes to use PSL as a communication medium between system's analysts and programmers. Since we do not believe a solution for the problem of generating code automatically from an extended PSL specification will be satisfactorily solved in the near future, we have attempted to train programmers to interpret correctly a PSL specification and produce reliable (verifiable) code for it. We have adopted a very simple approach, namely: define the semantics of PSL in terms of the programming language that will be used for program implementation. We have done this in COBOL [4].

In the process of constructing this semantic definition we had to deal with the current generality of PSL. Particularly, the variety of ways in which the system's dynamics can be expressed in PSL may lead to awkward equivalent constructions in COBOL. If we wish, as it was said before, to use the PSL system to specify the classical wide class of data processing systems, it makes sense to require that only well-constructed control structures be used in PSL. Well-constructed control structures are those that lead to COBOL program segments which can be considered structured in some sense [7, 8, 9]. In figure 9, we exemplify, how a well-constructed PSL control structure can be modelled based on the CASE statement, which is one of the classical one-entry-one-exit types of control structures

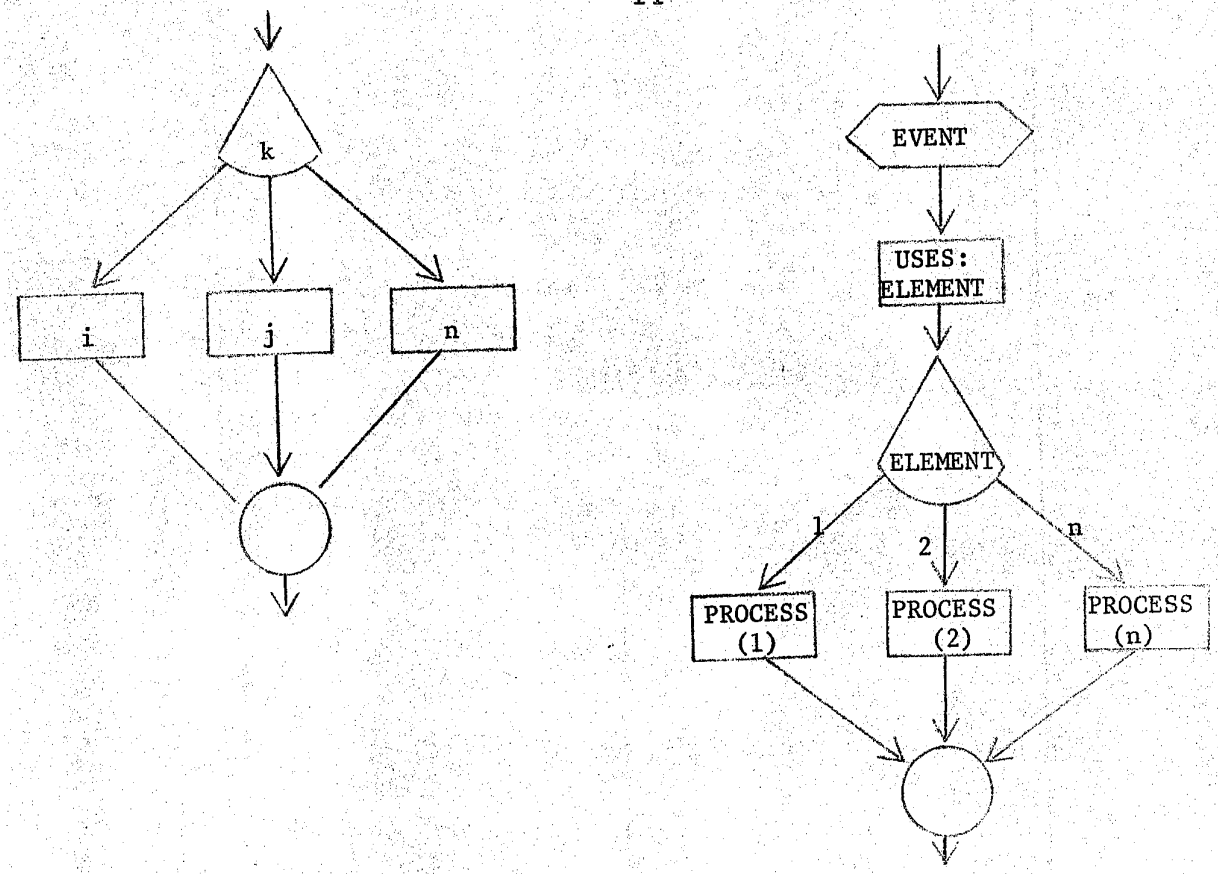


Fig. 9 The case structure

In Fig. 10 we present a PSL specification and the respective COBOL implementation of the structure.

a) PSL Spec. for Case

```
EVENT: event-name;
   USES element-name;
   TRIGGERS PROCESS (element-name);
PROCESS(1); name-of-process-1;
   DESCRIPTION;
        :
PROCESS(n); name-of-process-n;
   DESCRIPTION;
        :
```

b) COBOL implementation

```
PERFORM begin-selection THRU end-selection
        separate paragraph:
begin-selection
   COMPUTE K = PROCESS-1,
               PROCESS-2,
                  :
               PROCESS-n
           DEPENDING ON K
      GO TO not-found
PROCESS-1.step 1
   GO TO end-selection
PROCESS-2.step 2
        :
PROCESS-n.step n.
   GO TO end-selection
not-found.step 0.
end-selection
```

Fig. 10 PSL's Case Construction

In the process of attempting a rigorous COBOL statement of PSL's semantics we achieved, as a natural byproduct, the possibility of verifying systematically a COBOL program developed for a given PSL specification. In [4] a large number of verification rules are proposed, which follow the general interpreter equivalence approach, which is, for instance, described in [10]. Since the implementation language was also used for the semantic definition of the specification notation it is possible to precede the check of the dynamic behavior of the two level programs by extensive syntactic checks which try to ensure that the implementation contains all the objects and function descriptions prescribed in the specification.

5. CONCLUSIONS

We have tried to summarize a methodology which has been proposed to improve the benefits of using the PSL/PSA system for the development of data processing systems. Because of the very nature of the methodology (large number of rules) we decided to resort to many figures throughout the paper to suggest to the reader the nature of the available results. Experience, so far, has indicated that not only is it possible to better convince the user of the advantage of PSL, through the proposed methodology, but that is also suggests a number of improvements in the PSL/PSA system which are being investigated by our research group.

REFERENCES

[1] Teichroew, D., "PSL User's Manual", ISDOS Working Paper 98, University of Michigan, 1975.

[2] Teichroew, D and Bastarache, M., "PSA Output's, ISDOS Working Paper 99, University of Michigan, 1975.

[3] Delaroli, R., "Uma Técnica de Documentação Semi-Formal para Especificar Sistemas de Processamente de Informações", MSc. Dissertation, Pontifícia Universidade Católica do Rio de Janeiro, 1976.

[4] da Silva, V.G., "Validação de Sistemas de Programação", MSc Dissertation, Pontifícia Universidade Católica do Rio de Janeiro, 1976.

[5] Lynch, H., "ADS: A Technique in System Documentation", Database 1, 1, 1969.

[6] Nunamaker, J.F. et al, "Computer-Aided Analysis and Design of Information Systems, CACM, vol. 19, nº 12, 1976.

[7] Amorim, L.A.M., "Metodologia para a Síntese da Implementação de Sistemas de Programação a partir de uma Especificação Formal Bem Construida do Problema", MSc Dissertation, Pontifícia Universidade Católica do Rio de Janeiro, 1976.

[8] Gelder, A.V., "Structured Programming in Cobol: An Approach for Application Programmers", CACM, vol. 20, nº 1, 1977.

[9] McClure, C.L., "Structured Programming in COBOL", Sigplan Notices, vol. 10, nº 4, 1975.

[10] Mc Cowan, C.L., "An Inductive Proof Technique for Interpreter Equivalence", in Formal Semantics of Programming Languages, Prentice-Hall, Inc., 1972.

[11] Katzan, H., "System Design and Documentation - An Introduction to the Hipo Method", Van Nostrand Reinhold, 1976.