SVS, BORS, SVSi:
Three Strategies to relate Problem and Program Domains

Mario M. Berón
Universidad Nacional de San Luis
San Luis, Argentina
mberon@unsl.edu.ar

Maria João V. Pereira
Polytechnique Institute of Bragança
Bragança, Portugal
mjoao@ipb.pt

Nuno Oliveira
University of Minho
Braga, Portugal
nunooliveira@di.uminho.pt

Daniela da Cruz
University of Minho
Braga, Portugal
danieladacruz@di.uminho.pt

Abstract— Program Comprehension is improved if: i) the Problem and Program Domains can be related, and ii) this relation is shown in a suitable way to the programmer. Currently, there are few strategies for reaching this important goal because it is not easy to: i) Identify meaningful representations of the problem and program domains; and ii) Define a linking procedure.
This poster describes three strategies to overcome the difficulties mentioned above. These strategies use static and dynamic information and traditional compilation techniques for relating both domains.

Keywords- Program Comrephension; Problem Domain; Program Domain; Comprehension Strategies

I. INTRODUCTION

Program Comprehension is a discipline of Software Engineering aimed at helping the programmer to understand programs. In order to reach this goal, it is crucial to relate the two domains involved in computer-based problem solving: the Problem and Program Domains [2].

The first one refers to the system output. The second one is concerned with the program components used for producing the output.

When this inter-domain relationship is attained, the programmer can easily identify the software components interacting to perform a specific task. This capability makes easier the activities concerned with software maintenance and evolution but currently few program comprehension tools offer that.

This poster depicts three strategies to overcome these difficulties, they are: SVS (Simultaneous Visualization Strategy), BORS (Behavioral-Operational Relation Strategy) and SVSi (Simultaneous Visualization Strategy Improved). These strategies extract, from the code and from its execution, static and dynamic information needed to relate both domains. The strategies are briefly introduced below. Section 2 describes SVS (for details see [1]). Section 3 explains BORS (for details see [1]). Section 4 discusses a new approach, SVSi, not yet published.

II. SIMULTANEOUS VISUALIZATION STRATEGY

Simultaneous Visualization Strategy (SVS) requires dynamic information for linking the problem and program domains. In order to gather this information, the source code is instrumented. The instrumentation strategy inserts functions (called inspectors) at the beginning and at the end of each system function. The inspectors report the function name and other information that the programmer consider important for facilitating the comprehension process. The inspectors are implemented in other system called monitor.

After the instrumentation process some primitives for parallel execution are inserted in the system. The goal of this task is to allow the parallel execution of the system and the monitor.

When all the tasks previously mentioned are done, the system can be executed. The SVS' effect is shown in Figure 1.

The system (in this case xfig) is running and the monitor shows the functions used for building the xfig' interface. The reader can observe the direct relation between the Problem and Program Domains.

III. BEHAVIORAL-OPERATIONAL RELATION STRATEGY

Behavioral-Operational Relation Strategy (BORS) reach the same goal of SVS but using slight differences.

Like SVS, BORS uses dynamic information for carrying out its goal. This information is gathered using the same instrumentation scheme employed by SVS.

BORS has three steps clearly defined: i) Identify the problem domain objects and their interfaces; ii) Build a function execution tree (fe-Tree) and iii) Explain the interfaces retrieved in step i) using the tree build in step iii).

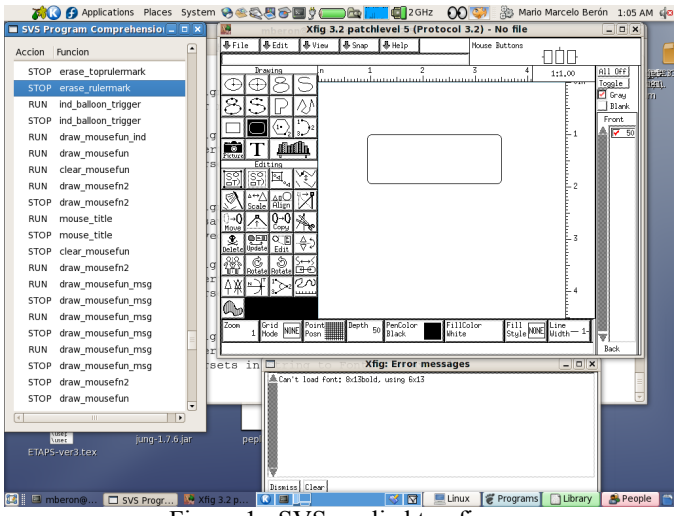

Figure 1: SVS applied to xfig

The first step is observational, the user sees the system output and then he tries to gather the interfaces using strategies such as: Grep Technique, Abstract Data Type Detection, etc. The result of this step is inserted in a list.

The second step uses both the dynamic information and a procedure for build a fe-Tree.

The third step applies a breath-first traversal on the fe-Tree. When one node in the list (see step one) matches with a node on the fe-tree, BORS reports: i) The path from the fe-Tree root to the node under analysis; and ii) The sub-tree of this node.

Figure 2 shows the BORS effect using as case study to agrep. The reader can observe the direct relation between problem and program domains.

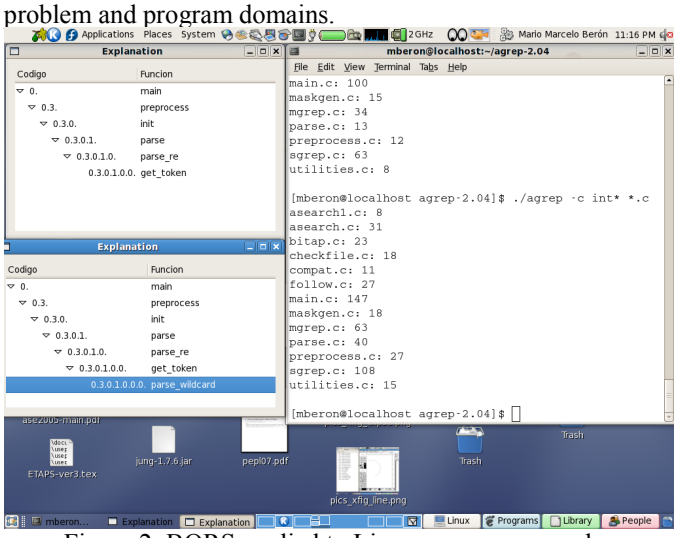

Figure 2: BORS applied to Linux agrep command

IV. SIMULTANEOUS VISUALIZATION STRATEGY IMPROVED

Simultaneous Visualization Strategy Improved (SVSi) was created for solving the problems found in SVS and BORS.

On one hand, SVS relate the problem and program domains, but it lacks of approaches for doing explanations.

On the other hand, BORS can proportionate explanations but it is semi-automatic (the first step is observational).

SVSi solves the problems mentioned above using the SVS output (functions used in runtime) as BORS input.

From section III is clear that BORS needs the object interfaces for providing some contextual information useful to catch the system functionality.

From section II (SVS strategy and its instrumentation scheme) is possible to know how the interfaces used to build the output objects can be retrieved.

With this idea in mind, two tasks need to be carried out for implementing SVSi. The first one is to modify SVS' monitor for recording the functions used for a specific system execution. The second one is to change the first step of BORS. In this case, the functions gathered by SVS are used BORS as its input. The implementation of this strategy is under development.

V. SYNTHESIS

In this poster three strategies for relating Problem and Program Domains are presented. The first one, SVS, can relate both domains when the system is running. The second one, BORS, creates this relation after the execution.

However, both approaches, SVS and BORS, have problems. On one hand, it is difficult with SVS to build explanations. On the other hand, BORS is semi-automatic. In order to overcome these problems, SVSi was proposed.

VI. REFERENCES

[1] M. Berón, P. Henriques, M. Varanda, and R. Uzal. "Inspección de Programas para Interconectar las Vistas Comportamental y Operacional para la Comprensión de Programas". Ph.D Thesis. Universidad Nacional de San Luis, San Luis, Argentina, 2009.

[2] R. Brook. "Using a behavioral theory of program comprehension in software engineering". Proceedings of the 3rd international conference on Software engineering, pages 196{201, 1978.