INKA : AN AUTOMATIC SOFTWARE TEST DATA GENERATOR FOR THE FULL COVERAGE OF TESTING CRITERIA

I. INTRODUCTION

Software is today the critical part of more and more systems on which depends the every day life of many people (for instance communication, transportation, business systems). A scrupulous test of these systems and in particular of the software part is thus an obligation before delivering them. Unfortunately, the difficulty and the cost of the test of software lead to put in operation to fragile systems.

In fact, software can be viewed as a set of individual components and software testing involves defining input test data, executing each component with the test data and finally checking the observed results against expected outputs. Among the strategies used for testing a component, structural testing is imperative to ensure a certain level of security. It consists in selecting test data such as every part of the source code of the components is executed at least once during the test process (would you accept to fly on board an aircraft for which the airborne software system has some code statements which are executable and have never been executed ?). The manual definition of such test data is a major time-consuming activity. For instance, an experiment made recently in our company showed that the manual definition of test data takes 1,5 hour in average for each component, since a classical software contains about 1000 components, this activity takes up an engineer during a complete year. In fact, constructing a test datum for which a part in the source code is executed is a non-obvious intellectual process : it requires to understand the conditions under which that part can be executed and to solve a kind of chinese puzzle to find the data respecting these conditions.

II. TECHNICAL VIEW

For almost 25 years, research works have been undertaken in order to discover the best way for automating the generation of software test data. However, so far there is no software testing tool on the market able to provide automatically test data such as every part of a program would be executed.

After 4 years of intensive research [1,2,3,4], and thanks to the use of the sound and now well-proved Constraint Programming techniques, we can propose a decisive solution (called InKa) to this problem. Constraint Programming replaces classical imperative code statements by constraints which are logical relations between different variables of a problem to be solved. CP gives then automatic and integrated methods to solve this problem.

InKa exploits this property by translating the source code (written in C or C++) of the component to be tested into a CP program. The selection of a statement in the source code leads to set up a constraint system on the input variables of the component. Then, InKa computes automatically the solutions of the resulting constraint system. It does this task for each executable statement of the component to find the test data ensuring a 100% coverage of its structure.

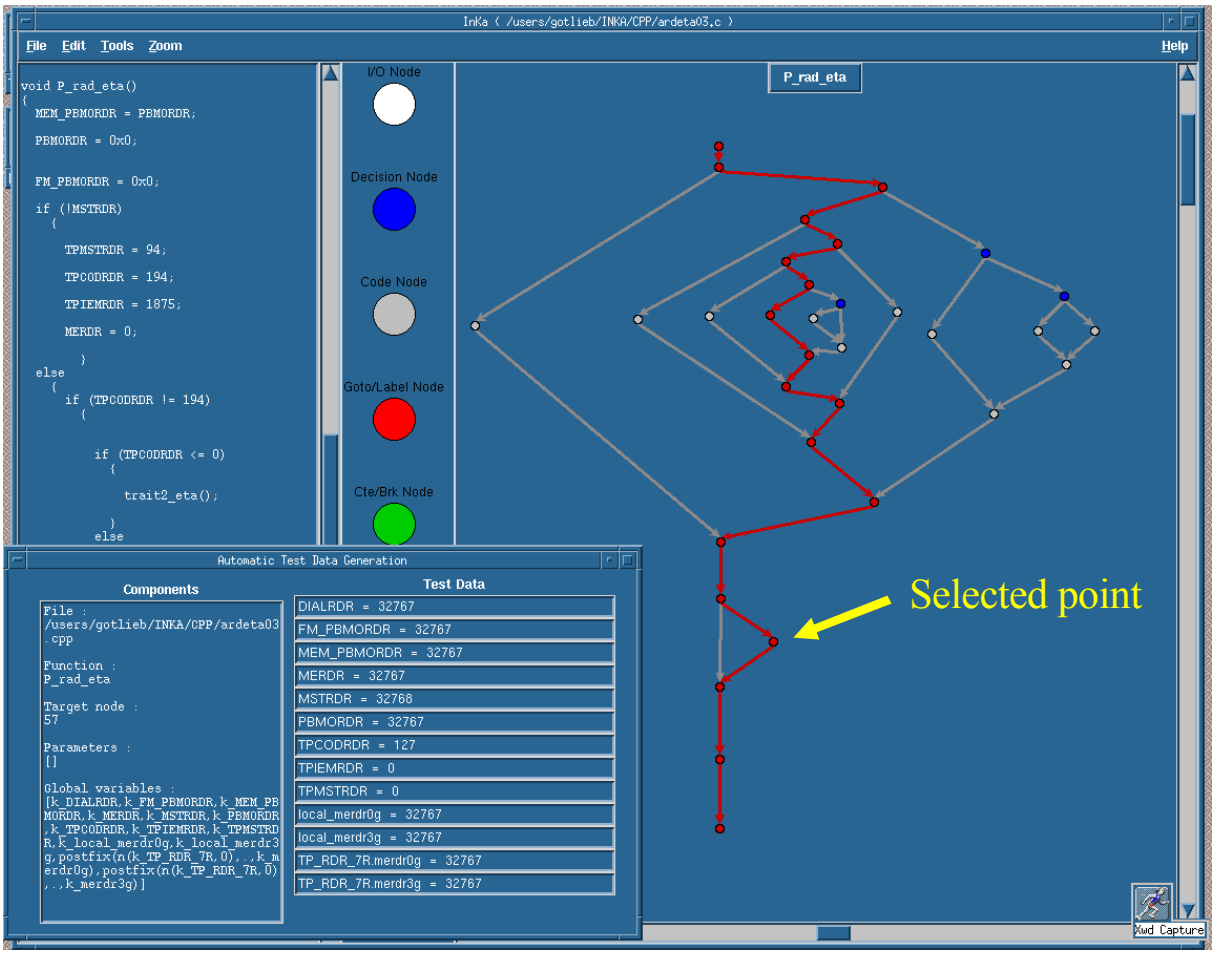
On another hand, InKa can sometimes detect when the resulting constraint system has no solution, in this case it exhibits a non-obvious default of the software under test : a non-executable part of the program (sometimes called dead code). As a result, InKa is the first tool able to detect such defaults in programs in the presence of complex constructions. Nevertheless, its main function remains automatic test data generation.

To validate this new approach, a prototype of InKa was developped and experimented. The results of this prototype confirmed our hopes :

- it achieved, with a set of restrictions on the statements of programs to be analysed, the complete process of automatic test data generation (inputs of InKa are the source code of a program and a selected point, output is a test datum which executes the selected point) ;
- it was used successfully both to analyse academics programs for which the automatic test data generation is known as being difficult and for real programs extracted from an airborne software system. In each case it achieved a complete coverage. Results are published in [3,4] ;
- its results were compared experimentally with random testing, which involves selecting randomly the test data and checking the observed coverage of programs. It has been shown that InKa overcomes random testing in quality (this is not surprising because random testing is a blind generation of test data and then it does not ensure the complete coverage of programs) but also in efficiency [3,4].

A snapshoot of the experimental prototype InKa is given below. A program written in the C language is shown on the left part of the window called “InKa(/users/gotlieb/inka/CPP/ardeta03.c)”. Its control graph, which is an abstract representation of the structure of the program has been built by InKa and it is shown on the right part of the same window. After the selection, by the user, of a point in the control graph (the one shown by the yellow arrow), a new window appeared, called “Automatic Test Data Generation”. An input test datum has been

automatically computed and it is given here : its values are : [DIALRDR = 32767,…, TP_RDR_7R.merdr3g = 32767]. The path corresponding to this test datum has been computed by InKa and it is shown in red on the control graph. So, the user can see what are the points already covered and he can select other points, not yet covered, in order to obtain test data ensuring a 100% coverage of program. Of course, all that can be carried out automatically.



III. HISTORICAL AND INDUSTRIAL PERSPECTIVES

THALES Systèmes Aéroportés (ex. DASSAULT ELECTRONIQUE) experienced software testing for a long time, either as applying a testing process (for example, testing the airborne software system of the RAFALE aircraft) or developing software testing tools (for example, the development and marketing of DEVISOR).

In 1995, we studied the possible improvements of the testing process applied in our Company and we observed that it would be interesting to automate the selection of input test data [5]. We understood that such a process improvement wouldn't have to modify the current testing practices (by using formal methods for example). In parallel, our knowledge of Constraint Programming techniques and the previous works we have done in this area [6,7,8] have led us to explore the use of these techniques to treat the difficult problems of automatic test data generation.

Research works on this problem have been initiated with the I3S laboratory of the University of Nice-Sophia Antipolis. In 1996, we have proposed a new method of test data generation based on the use of Constraint Programming techniques [1]. Setting up an experimental prototype tool (InKa) which demonstrates the technical feasibility of our approach has yield a first international publication in 1998 [2]. Successful results obtained on programs extracted from a real airborne software system (as shown on the Phd thesis which describes these works [3]) have led us to decide the industrialisation of the prototype.

The development of the product will take two years and will start in january 2001. The industrialisation of InKa will be both funded by the French Ministry of Industry in the frame of the RNTL program and by internal investments. THALES Systèmes Aéroportés will work in collaboration with AXLOG Ingénierie for the development, the packaging and the marketing plan of the product. We will work also with the I3S laboratory of the University of Nice-Sophia Antipolis, which will drive the Research in collaboration with two other laboratories on some remaining research points to solve : the LIFC laboratory of University of Franche-Comté and the LSR-IMAG laboratory of the University of Grenoble. These remaining points have already been studied but require more research works [3].

The outgoing of two versions of InKa is planned. The first version V0 will be devoted to the unit testing process. The works consist in the industrialisation of the experimental prototype as it is. The second version V1 will include the solutions of the remaining research points provided by the laboratories. This version will be devoted to both unit and integration testing.

IV. CONCLUSION

InKa will be an industrial product in a near future (2002). Its design will made it independent from any other software testing tool in order to be integrated into any software development chain. InKa will not require a change in the current development practices. It will be usable at the unit level of software testing (testing alone each elementary component of the program) but also at the integration level (testing all together the components of the program).

Using InKa will reduce dramatically the cost of software testing, because the test data will be automatically provided by the product and they will be directly used as inputs by the testing tool of the development chain. Furthermore, it will detect some defaults in the program under test, which are impossible to detect with any other tool. It will be able to generate test data ensuring that the entire executable part of the program (100%) is covered, improving so the quality of software. Privilegied domains includes civil and military airborne software systems, real-time softwares, nuclear energy control systems, software railway systems…

[1] **Arnaud Gotlieb, Franck Calvet, Michel Rueher**. *Génération Automatique de Cas de Test : une approche Programmation Logique par Contraintes*. Published in "Génie Logiciel", Nov. 1996, EC2.

[2] **Arnaud Gotlieb, Bernard Botella, Michel Rueher**. *Automatic Test Data Generation using Constraint Solving Techniques*. Software Engineering Notes, 23 (2) :53-62, Mar. 1998, ACM.

[3] **Arnaud Gotlieb**. *Structural and Automatic Test Data Generation by using Constraint Logic Programming* Phd Thesis, University of Nice-Sophia Antipolis, Jan. 2000, in French.

[4] **Arnaud Gotlieb, Bernard Botella, Michel Rueher**. *A CLP framework for computing structural test data*. In International Proceedings of Computational Logic, London, Jul. 2000. Available at http://www.essi.fr/~rueher

[5] **Antoine d'Hennin**. *Etude du comportement d'experts en test de logiciel en vue d'automatiser la génération de test*. Rapport du DEA Automatique Industrielle et Humaine, Université de Valenciennes et du Hainaut-Cambresy. Juil. 1995.

[6] **Olivier Lhomme**. *Consistency techniques for Numeric CSPs*. In proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, 1993.

[7] **Olivier Lhomme, Arnaud Gotlieb, Michel Rueher, Patrick Taillibert**. *Boosting the Interval Narrowing Algorihtm.* In proceedings of Joint International Conference and Symposium on Logic Programming, pages 378-392, Bonn, sept. 1996. MIT Press.

[8] **Olivier Lhomme, Arnaud Gotlieb, Michel Rueher**. *Dynamic Optimization of Interval Narrowing Algorithms*. Journal of Logic Programming, 37:164-182, 1998