Debugging in consumer-programming oriented environments

Zvonimir Pavlic*, Tomislav Lugaric** and Marin Silic *
* University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
Consumer Computing Laboratory, Zagreb, Croatia
**Laboratory for Underwater Systems and Technologies
zvonimir.pavlic@fer.hr

Abstract - Computer consumers are the largest group of computer users, many of which are highly creative and experts in their area. Despite the fact they have no formal education in computer programming, they want to express their creativity and develop their own applications which will satisfy their needs. Nowadays, consumers can build their own personalized software artifacts using Geppeto (Widget Parallel Programming Tool) by building personalized workflows and dataflows over widgets, small standalone Web applications. However, computer consumers are prone to making mistakes while programming, which results in bugs in their applications. Consumers require assistance of skilled programmers in order to build dependable and error-free applications. This paper discusses new debugging methods based on professional debugging techniques, which will be understandable to average consumer. These methods will allow consumers controlled execution of their applications in order to find and remove bugs. Methods described in this paper are suitable for debugging consumer applications in a widget-oriented consumer programming environment like Geppeto, and include animated step-by-step execution of the consumer's application, adding breakpoints within widget composition and introducing interactive backtracking in order to detect erroneous widget.

I. INTRODUCTION

Computer consumers are the widest group of computer users, which have no formal education in computer programming [1,2]. Many of them are highly creative and experts in their area of interest. They are keen to use professional and personalized applications which satisfy their requirements by their functionality and ease of use. In order to achieve the best quality of experience, as well as express their creativity, consumers build their own applications out of predefined software components made by professionals.

The most common form of software artifacts suitable for consumer's usage are World Wide Web applications [1]. Widgets are small standalone applications displayed in a Web browser. Widgets are equipped with graphical user interface for the interaction with the background process, which is usually a Web based source.

Consumers select a set of widgets relevant to their field of interest and build personalized data flows between widgets by interconnecting their graphical user interfaces. In order to develop complex personalized applications, consumers have to manually interact with multiple widgets, which is impractical. Consumers can use Geppeto (*Widget Parallel Programming Tool*), for automating widget interconnection [1,2,3,4]. Geppeto is a consumer-oriented framework for building workflows of consumer's applications, developed at the University of Zagreb, Faculty of Electrical Engineering and Computing (FER). Applications are built by using widgets as building blocks.

Lacking sufficient education, average consumers, unlike the professional programmers, do not understand the main concepts of programming crucial for development of new applications. Therefore, consumers are prone to making mistakes in their application, even more than the professional programmers, introducing bugs and errors in their applications. While professional programmers have many debugging tools and methods available, none of them are suitable for consumers and their level of knowledge.

This paper discusses new debugging methods based on professional debugging techniques which will be understandable to the average consumer.

The paper is organized as follows. Debugging methods available in professional programming environments are described in Section 2. Section 3 provides an overview of existing debugging methods in various consumer and end-user oriented systems. Section 4 proposes new debugging methods applicable to widget-oriented consumer programming environment. Section 5 gives a short example of consumer's debugging.

II. DEBUGGING

The programming process is divisible into several principal stages: formulate problem, generate plan, code, debug, and verify [5]. Programmers, even most experienced ones, write programs that contain errors. The first indication that a program is incorrect is usually an externally visible symptom, such the wrong value being printed or the system encountering a fatal problem. This externally visible symptom is called a *failure*. A failure is caused by an erroneous internal state (called an *error*) of the program. This error could be an incorrect value for a variable or the program executing in the wrong place. An error state is usually preceded by another error state. This chain of errors can be followed back to the cause. The

cause of the initial error is an algorithmic fault in the program. In debugging, these faults are called *bugs* [6].

Debugging is the process of identifying the root cause of an error and correcting it [7].It contrasts with testing, which is the process of detecting the error initially. Debugging is a difficult job because the programmer has little guidance in locating the bugs. To locate a bug that caused an error, the programmer must think about the causal relationships between events in a program's execution. There is usually an interval between the time when a bug first affects the program behavior and when the programmer notices an error caused by the bug. This interval makes it difficult for the programmer to locate the bug. On some projects, debugging occupies as much as 50 percent of the total development time. For most programmers, debugging is the hardest part of programming [5].

Debugging is a process that has several important and unavoidable steps. First step is to reproduce an error by bringing the program into a previously encountered erroneous state. This task can be very demanding, because some programs do not have reproducible and deterministic behavior, such as highly parallel programs. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. This simplification can be done by using divide-and-conquer approach. Programmers than use different debugging techniques to determine the location of the bug [8].

Tracing is debugging technique of watching (live or recorded) trace statements, or print statements that indicate the flow of process execution. Remote debugging is the process of debugging a program running on a system different than the debugger. Post-mortem debugging is debugging of the program after it has already crashed. Related techniques often include various tracing techniques and/or analysis of memory dump of the crashed process. The dump of the process could be obtained automatically by the system or by a programmer-inserted instruction [8].

Modern integrated development tools provide developers with debuggers – specialized tools used in debugging process. Debuggers are used to examine program states, to control execution of a program and to track down the origin of the problem in the code. During debugging, it is very important to change one thing at a time, and to keep an audit trail of changes made in the code. After the bug has been removed from the code, the program has to be tested again, in order to prevent new bugs inserted in the code during the debugging.

III. DEBUGGING IN CONSUMER-ORIENTED PROGRAMMING

Consumer-oriented paradigm has become the most common form of programming in use today [9], but there has been little investigation into the dependability of the programs that consumers create. This is problematic because the dependability of these programs can be very important. Errors in consumer applications, such as formula errors in spreadsheets, have cost millions of dollars [10]. This problem has been recognized and some solutions have been proposed.

"*Interrogative debugging*" technique has been presented for the event-based programming environment Alice. Consumers pose questions in the form of *"Why did…"* or *"Why didn't…"* that the system answers by displaying visualizations of the program. This work builds on their model of programming errors [11], which classifies errors and their causes. Other strategies are *statistical outlier finding* [12] and *anomaly detection* [13], which use statistical analysis and interactive techniques to direct consumer programmers' attention to potentially problematic areas during automation tasks.

Since the spreadsheet paradigm is very popular amongst consumers considerable efforts nave been put into work supporting program comprehension and debugging by end users in the spreadsheet paradigm. This includes devices to aid spreadsheet users in dataflow visualization and editing tasks. Similar groups of cells are recognized and shaded based upon formula similarity, and are then connected with arrows to show dataflow. This technique builds upon the Arrow Tool, a dataflow visualization device [14].

In What You See Is What You Test (WYSIWYT) methodology [9, 15], consumer can test a spreadsheet incrementally as it is being developed by simply validating any value as correct at any point in the process. Behind the scenes, these validations are used to measure the quality of testing in terms of a test-adequacy criterion. These measurements are then projected to the user via several different visual devices, to help them direct their testing activities.

Consumer mashup programming tools, like Marmite and Vegemite, implement execution control elements, which can help consumer in the debugging process [16, 17, 18]. Every mashup element has start, stop and play buttons, which consumer can use in controlling the workflow of his mashup. Additionally, Marmite and Vegemite have separate window where consumer can see the data set changes after applying each building block, and therefore observe dependencies between data and control flow.

Study of cognitive processes during debugging revealed that debuggers for consumer programmers should have the following features [9,19]:

- Animated view of execution
- Display of action data correlated with focus in the action history
- Incrementally-generated history of action execution
- Access from action values to the action history that processed them
- Access from action history to the corresponding action data

IV. DEBUGGING IN WIDGET-ORIENTED CONSUMER PROGRAMMING ENVIRONMENT

In widget-oriented consumer programming environment like Geppeto, consumers create their own personalized widgets by combining previously developed

Fig. 1. Controller

widgets into one composite widget [2]. Consumer can choose the set of widgets which provides the required set of functionalities for consumer application. Widgets are loaded into Geppeto container by entering widget's URL into container interface [3]. These widgets are considered source widgets for the composite widget. After loading the chosen widgets, consumer adds a programmable widget to container.

Programming in Geppeto consists of two steps: defining the user interface of consumer's composite widget and defining the composite widgets logic by building a personalized data flow through selected widgets [20].

Designing the user interface of a composite widget is done by adding graphical user elements from one or more other widgets that consumer has chosen as source widgets. Adding elements is done via the right click menu which can be brought up when the mouse pointer is over a user interface element.

When programming the data flow and the logic behind the composite widget, actions are specified using the right-click menu. Defining a sequence of actions is done by selecting "When clicked" action on an element. Actions of source widgets are defined by option "click" done on action buttons of source widget. Consumer can use input elements to type in text. Communication between source widgets is done by copying output of one widget and pasting it to input of another widget. All sequences the user generates can be viewed and reorganized in a table which is stored together with the generated composite widget.

Consumers understand the level of abstraction of graphical user interfaces and data flows, as well as the concept of widgets and widgets composition. Therefore, debugging tools should be exposed to consumers as widgets as well.

Figure 1 shows the *"Controller"* widget. It enables controlled execution of consumer's application through an animated step-by-step execution. When clicked on the *"Add Watch"* button, a drop down menu is displayed, where consumer can choose between several composite gadgets loaded in the container. Consumer can enter how many actions in a sequence will represent one step of the controller. The *"Play"* button executes one step. Sequence of actions is animated by highlighting the GUI element of the source gadget which is the object of the current action. Consumer can now observe the correlation between data changes and actions that processes that data in a controlled environment,.

Figure 2 shows the *"Debugger"* widget. By clicking on the *"Add Break"* button, consumer can add a breakpoint in the composite widget control flow. A table of actions is displayed, where consumer can choose one action and add a breakpoint after it. After defining a breakpoint, a new gadget, *"Breakpoint"*, shown on Figure 3, is inserted into the consumer application. GUI of the *"Breakpoint"* widget is defined by the GUI element of the source widget affected by the action following the breakpoint. In addition, the *"Play"* button is added, so that consumer can continue execution of his application after the breakpoint.

By clicking on the *"Trace"* button of the *"Debugger"* widget, a table of actions is shown. Consumer can backtrack an erroneous widget by building dependencies between widgets. Consumer can grade each output value of a widget as correct one (√) or incorrect one (X). When an output of a widget is marked as incorrect, the system automatically highlights widgets responsible for calculation of the incorrect value, like the widgets that give input values to the observed widget. This interactive process helps the consumer to concentrate only on widgets that can possibly be faulty, and guides him through the debugging process.

V. CONSUMER DEBUGGING EXAMPLE

This section describes the simple consumer-programmed application, where previously described debugging techniques will be demonstrated. The goal of this application is to sum two numbers, *A* and *B*.

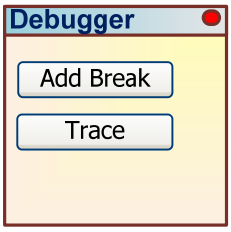


Fig. 2. Debugger

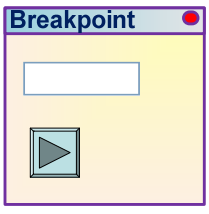


Fig. 3.Breakpoint

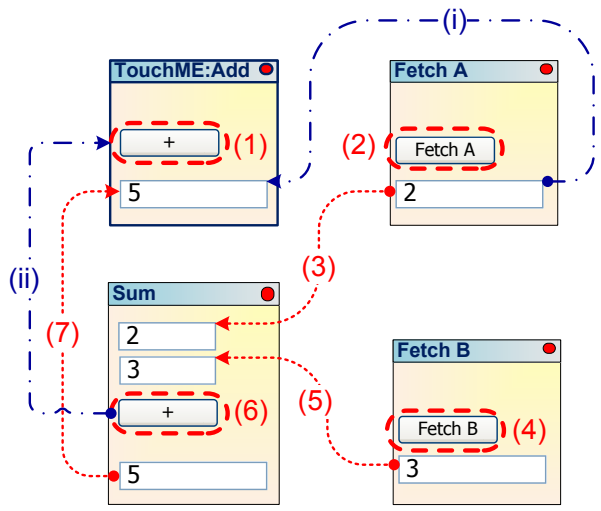


Fig.4. Consumer application

Consumer selects source widgets, *"Fetch A"*, *"Fetch B"* and *"Sum"* and loads them into the container. Consumer than creates a programmable composite "TouchMe" widget called *"Add"* and defines widget's GUI and behavior using Geppeto. Graphical user interface (GUI) of the composite widget is defined. Consumer adds the *"+"* button from the *"Sum"* widget (i) and the textbox element from the *"Fetch A"* widget (ii).

To build the workflow logic, the consumer selects *"+"* button at the composite widget and defines set of actions after selecting *"When Clicked"* option in the Geppeto drop-down menu (1). Consumer clicks on the *"Fetch A"* button of the *"Fetch A"* widget in order to fetch value of variable *A* (2). He copies value of variable *A* to the first input field of the widget *"Sum"* (3). Then he clicks on the *"Fetch B"* button of the *"Fetch B"* widget to get the value of variable *B* (4) and copies it to the second input field of the *"Sum"* widget (5). After clicking on the *"+"* button, the value *A+B* is displayed in the output field of the *"Sum"* widget (6). The consumer copies that value from the "Sum" widget to the output data field of the composite "Add" widget (7). The table of consumer-programmed actions that define data and control flow of the composite widget can be seen in Table 1.

TABLE 1. TABLE OF ACTIONS

Click fetchA @ fetchA
Copy A @ FetchA to A @ Sum
Click fetchB @ fetchB
Copy B @ FetchB to B @ Sum
Click + @ Sum
Copy S @ Sum to res @ Add

In order to control the execution of his application, consumer adds new widget, *"Controller"* to his application. The Consumer than adds watch to the composite "Add" widget via the user interface of the *"Controller"* widget, and sets the step on 1 action.

When consumer clicks on the play button, the first action will be executed. Controller widget will click on the *"Fetch A"* button on the *"Fetch A"* widget, which will be highlighted (1). After the second click on the play button, the output box of the *"Fetch A"* widget and the first input box of the *"Sum"* widget will be highlighted (2), and the value of variable A will be copied from widget *"Fetch A"* into the widget *"Sum"* (3). On the following consumer's clicks on the *"Play"* button, actions (4) to (7) from the table of actions will be executed respectively with highlighting of the affected graphical user interface elements.

To add a breakpoint in his application, the consumer uses a *"Debugger"* widget, displayed in the Figure 5. When clicked on *"Add Break"* button, a table of actions is displayed (i). The consumer can see list of actions and chooses to add a breakpoint after the first action (ii). After defining a breakpoint via the *"Add Break"* button, a new gadget, *"Breakpoint"* is inserted into the consumer application. GUI of the *"Breakpoint"* widget consists of the input field of the "Sum" which is the destination of copy action where breakpoint was added (iii).

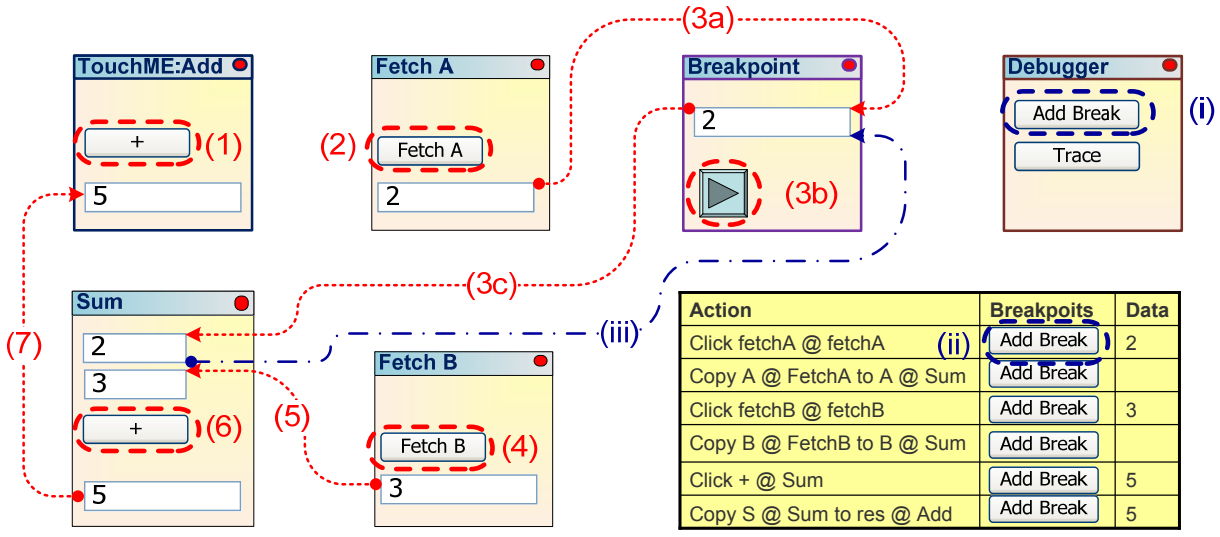
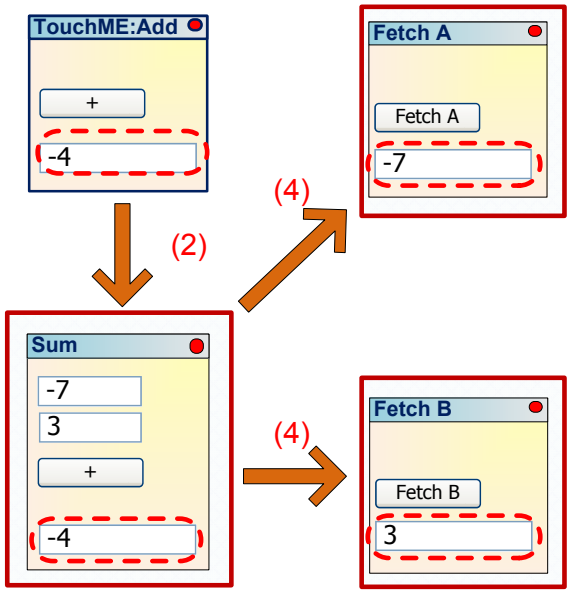


Fig. 5. Adding breakpoint



Action	Corr	Data	
Click fetchA @ fetchA	?	-7	
Copy A @ FetchA to A @ Sum	✗		(6)
Click fetchB @ fetchB	✓	3	(5)
Copy B @ FetchB to B @ Sum	?		
Click + @ Sum	✗	-4	(3)
Copy S @ Sum to res @ Add	✗	-4	(1)

Fig. 6. Backtrtacking erroneous widget

Now, the consumer starts his application by clicking on the *"Add"* button of composite *"Add"* widget. Actions will be executed sequentially as displayed in the table of actions until the action (2) is finished. Action (3a) will copy the value from widget *"Fetch A"* into the textbox of *"Breakpoint"* widget and the execution will be halted. Consumer can now see the values of his data sets after actions prior to break point. After clicking on the *"Play"* button (3b), value from breakpoint widget will be copied into the first input field of the *"Sum"* widget (3c). The execution will continue from action (4) until the last action in the table of actions.

In case the consumer's application didn't produce the expected result, consumer must find out which widget from his composition is responsible for that error, in order to debug his application. The *"Trace"* option of the *"Debugger"* widget can be used backtrack an erroneous widget by examining correlations between widget actions and data set changes. Tracing is shown in Figure 6.

The consumer detected that his application, composite widget *"Add"* produces the wrong result. Variable A and B should be positive, yet the result of their summation is negative. The consumer marks the output value of the *"Add"* widget as incorrect, *"X"* (1). The debugger automatically selects the widget *"Sum"*, that is responsible for the output of widget *"Add"* (2). The consumer now examines actions of widget *"Sum"*. Addition is marked as incorrect, *"X"*, (3). The debugger selects widgets "Fetch A" and *"Fetch B"* which provides an input values to *"Sum"* widget (4). The consumer examines action of selected widgets. He marks action at *"Fetch B"* as correct *"√"* (5), and action at *"Fetch A"* as incorrect *"X"* (6). Therefore, the bug has been detected, and the widget *"Fetch A"* is labeled as faulty. The consumer must reprogram his application, and replace the faulty widget *"Fetch A"* with the correct one.

VI. Conclusion

This article addresses the problem of debugging in the consumer-programming oriented environments. Consumers are prone to making mistakes in their applications, and need assistance of experienced programmers in order to write reliable software. However, because consumers are different from professional programmers in background, motivation, and interest, their needs cannot be addressed by simply repackaging techniques and tools developed for professional software engineers. Debugging methods available to professional programmers are not suitable for consumers, since they do not have necessary knowledge of programming process.

New debugging methods based on professional debugging methods are proposed, that are appropriate to average consumers and that will motivate them to debug their applications. These methods are designed for debugging consumer applications in a widget-oriented consumer programming environment, and include animated step-by-step execution of the consumer's application, adding breakpoints and interactive detecting erroneous widget.

Acknowlegment

The authors acknowledge the support of the Ministry of Science, Education, and Sports of the Republic of Croatia through research project „Computing Environments for Ubiquitous Distributed Systems“ (036-0362980-1921) as well as Google, Inc. for the Google Research Award project "End-User Tool for Widget Composition" and "CURE – Developing Croatian Underwater Robotics Research Potential" SP-4 Capacities (call FP7–REGPOT–2008–1) under Grant Agreement Number: 229553. Furthermore, many thanks to Sinisa Srbljic, Miroslav Popovic and Dejan Skvorc from Faculty

of Electrical Engineering and Computing at University of Zagreb for their help with preparing this manuscript.

REFERENCES

[1] Škvorc, D.; Programiranje prilagođeno potrošaču, PhD thesis, University of Zagreb, Faculty of Electrical Engineering and Computing

[2] Srbljić S., Škvorc, D. Skrobo, D.; Widget-Oriented Consumer Programming, Automatika 50(2009), 3-4, str 252-264

[3] Geppeto home page, http://www.ris.fer.hr/, 27.01.2012.

[4] Popović, M.; Consumer Program Synchronisation. PhD thesis, University of Zagreb, Faculty of Electrical Engineering and Computing

[5] Gould, J. D.; Some psychological evidence on how people debug computer programs, International Journal of Man-Machine Studies, Volume 7, Issue 2, March 1975, Pages 151–170, IN1–IN2, 171–182

[6] Miller, B. P., Choi, J.-D.; A Mechanism for Efficient Debugging of Parallel Programs, PLDI '88 Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation

[7] McConnell, S. C.; Code Complete: Book and Online Course Bundle, 2nd edition, C.B.Learning , United Kingdom ©2010

[8] Agans, D. J.; Debugging: The Nine Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems, Amacom, 2002.

[9] Burnett, M., Cook, C., Rothermel, G.; End-user software engineering, Communications of the ACM - End-user development: tools that empower users to create their own software solutions, Volume 47 Issue 9, September 2004

[10] Ko, A.J., Myers, B.A.; Designing the Whyline: A Debugging Interface for Asking Questions about Program Failures, Proc. ACM Conf. Human Factors Computing Systems, 2004, 151-158.

[11] Harrison, W.; From the Editor: The Dangers of End-User Programming , IEEE Software, July-Aug. 2004, Volume: 21 Issue: 4, pages: 5 - 7

[12] Miller, R.C., Myers, B.A.; Outlier Finding: Focusing User Attention on Possible Errors, Proc. ACM Symp. User Interface Soft. Technology, 2001, 81-90.

[13] O. Raz, P. Koopman, and M. Shaw, "Semantic Anomaly Detection in Online Data Sources", Proc. Int. Conf. Soft. Eng., 2002, 302-312.

[14] T. Igarashi, J.D. Mackinlay, B.-W. Chang, and P.T. Zellweger, "Fluid Visualization of Spreadsheet Structures", Proc. IEEE Symp. Visual Langs., 1998, 118-125.

[15] Rothermel, G., Li, L., DuPuis, C., Burnett, M.; What You See Is What You Test: A Methodology for Testing Form-Based Visual Programs, Proceedings of the 1998 International Conference on Software Engineering, 19-25 Apr 1998.

[16] Wong, J.; Marmite: Towards End-User Programming for the Web, IEEE Symposium on Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007., 23-27 Sept. 2007, On page(s): 270 - 271

[17] Wong, J., Hong, J.; Marmite: end-user programming for the web, Proceeding CHI EA '06 CHI '06 extended abstracts on Human factors in computing systems

[18] Lin, J., Wong, J., Nichols, J., Cypher, A., Lau, T.A.; End-user programming of mashups with vegemite, IUI '09: Proceedings of the 14th international conference on Intelligent user interfaces, ACM New York, NY, USA 2009

[19] Lieberman, H., Wagner, E.; End-user debugging for e-commerce, IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces, ACM New York, NY, USA 2003

[20] Pavlic, Z.; Lugaric, T.; Srbljic, S.; Consumer-oriented programming application for statistical processing, MIPRO, 2011 Proceedings of the 34th International Convention, 23-27 May 2011, On pages: 702 - 706