A multiplatform interpreter to introduce structured and concurrent programming

Beatriz Depetris[1], Guillermo Feierherd[1], Daniel Aguil Mallea[1], Germán Tejero[1]

[1] Universidad Nacional de Tierra del Fuego, Instituto de Desarrollo Económico e Innovación,
Hipólito Irigoyen 880, 9410 Ushuaia, Tierra del Fuego, Argentina
{bdepetris, gfeierherd, daguilmallea, gtejero}@untdf.edu.ar

Abstract. The process of teaching and learning computer programming has always been a challenge for students and teachers. Throughout time, the challenge has become tougher because now the concepts related to concurrent programming must be added to the traditional concepts of programming. The tools that show the performance of algorithms have been of great help, although they must be used carefully. This article describes the development and use of an update of the Visual Da Vinci (an environment used in different institutions to introduce computer programming) and discusses the reasons why programming (and software design in general) usually arises issues. It also justifies the current importance of concurrent programming and the difficulties it adds to traditional programming. Further on in the text, the Concurrent Da Vinci is analysed and commented on, showing an example of how it is used to solve a classical problem of concurrency.

Keywords: teaching computer programming, teaching concurrent programming, algorithm visualization.

1   Introduction.

Having a flair for managing abstract ideas is a distinctive characteristic of good computing professionals. It is an ability made up of two complementary elements: the ability to simplify, removing unnecessary details, and the ability to derive generalizations that highlight the common and essential aspects of a group of specific cases.

This ability is necessary because software is essentially abstract, therefore its design and development have to do mainly with abstractions. As Devlin states, *"Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner."* [1]

Along the same line, when Kramer wonders *"why is it that some software engineers and computer scientists are able to produce clear, elegant designs and programs, while others cannot?"*, he reaches the conclusion that *"the key lies in*

abstraction: The ability to perform abstract thinking and to exhibit abstraction skills." [2]

According to Piaget, humans develop the abilities related to abstraction at the fourth (and last) stage of cognitive development: the formal operational stage, which starts approximately at 12 years old. Nevertheless, reality shows that many university students (at least among the ones that chose courses related to informatics) have a low development of these abilities.

Piaget's theory backs up the assertion of Rutherford and Ahlgren, quoted by Dann and Cooper in their article about Alice: *"students' learning progression is usually from the concrete to the abstract. Young people can learn most readily about things that are tangible and directly accessible to their senses—visual, auditory, tactile, and kinesthetic. With experience, they grow in their ability to understand abstract concepts, manipulate symbols, reason logically, and generalize. These skills develop slowly, however, and the dependence of most people on concrete examples of new ideas persists throughout life. Concrete experiences are most effective in learning when they occur in the context of some relevant conceptual structure."* [3]

1.1   Teaching Computer Programming

Given that humans understand abstract ideas basing on concrete ideas, tools aimed for the introduction of young people to programming have been created and used for some time. The common characteristic of these tools is the visualization of the algorithms' execution. These tools are usually used in introductory courses to computer programming of the university courses that deal with informatics (CS1 and CS2). Nevertheless, because of the relevance computing thought has gained lately, in some cases the tools are used with under-age students, or even with students that are not planning to venture into informatic courses. While some of these tools have specific purposes (for example, visualizing the action on trees), others, even if they can be of use for people with different ages and interests, are more general (like Alice, Greenfoot and Scratch). [4]

At the ex Ushuaia headquarters of the Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB), currently part of the newly created Universidad Nacional de Tierra del Fuego, we have been using Visual DaVinci (VDV) for several years. The environment VDV is a tool designed and developed in the Instituto de Investigación en Informática LIDI of the Universidad Nacional de La Plata. This institution started to use it, and it was later implemented in other National Universities. [5]

One of the main characteristics of the Visual DaVinci language is that it can receive instructions in Spanish. In order to be able to visualize the execution, it uses a city made compose of streets (horizontal) and avenues (vertical), a robot (that can move through the city) and objects (flowers, papers and obstacles), located in the intersection of streets and avenues, with which the robot can interact.

Before executing a program, the location of flowers, papers and obstacles can be determined. During the execution, the robot (Lubo-I), which has a bag to take flowers and papers, moves along the city obeying a group of primitives (*mover, derecha,*

tomarFlor, *depositarPapel*, etc.) Besides, there is access to certain variables of the system (*HayFlorEnLaEsquina*, *HayPapelEnLaBolsa*, etc.) [6]

Compared to any other visual programming language available in the market, Visual DaVinci is a limited tool, but it is precisely this characteristic which helps avoiding the distraction that a great number of possibilities causes, and diminishing the time it takes to learn how to use the enviroment. Thanks to this students can focus on learning to design algorithms and express them in a short period of time. Obviously, these limitations restrict both the possible data and the algorithms that can be developed and visualized.

Finally it should be noted that Visual DaVinci uses a syntax similar to Pascal, which facilitates the task in that courses of programming and algorithmic that use it as a language.

1.2   Previous Experiences at the Ushuaia Headquarters

VDV has been used at the Ushuaia Headquarters since 1999. At first, it was used in an optional course of Expression of Problems and Algorithms that used to take place simultaneously with the Maths Leveling Course, which is required to start any course at the Faculty of Engeneering of the UNPSJB.

The optional course had two basic aims. First, to work as a vocational guidance for the applicants to the course. Second, to start working with some contents of Algorithms and Programming (subjects of the second semester of the first year), allowing the concepts to sink in during the first semester.

The results of the experiences were most satisfactory: the failure level in Algorithms and Programming dropped among the students that had taken the optional course [7]. As a consequence, in the new syllabus implemented since 2010, the subject Expression of Problems and Algorithms was added to the first semester of the first year. At first, the subject works with VDV, and then starts with Pascal.

1.3   Concurrent Programming

Concurrency is a characteristic of some problems which allows them to be transformed into less serious sub-problems that, to a certain extent, can be solved simultaneously. However, given that these solutions are partial collaborations to the general solution, they are not completely independent. Depending on the problem, they have to *share resources* and *synchronize.* When this technique is applied to problems that must be solved by computers, it is called concurrent programming.

It is important to point out that concurrent programming does not mean *real simultaneity.* In fact, if there is one processor only, the simultaneous execution is only apparent. Besides, when the execution of the programs that solve partial solutions can take place really simultaneously, using a group of machines (distributed system),a machine with many processors (multicore), or some combination of both options, we prefer to call it parallel programming.

The construction of operating systems was a field where concurrency techniques sparked great interest and experienced significant improvements. As a consequence, in many universities the introduction to concurrent programming is usually a part of the subject that deals with the different aspects of Operating Systems. Afterwards, in many cases, both concurrent and parallel programming, receive little further treatment.

However, in recent times, hardware developments that took the ability of multiprocessing to cheaper devices generated an added interest in this way of solving problems. In the case of our syllabus, this has been reflected in some changes made to the syllabus, implemented since 2010. Some of the changes are the transformation of the concurrency topics of the subject Operating Systems into a new subject (Introduction to Concurrency) and the creation of curricular spaces specifically devoted to the issue (Parallel Systems-S).

To the already mentioned issues of traditional programming, concurrent programming adds new ones. Some of them are the need to establish a communication between the processes that contribute to the solution, the control of the access to shared resources and synchronization. However, we think that the greatest difficulty comes up when checking the processes' correctness, as a consequence of *nondeterminism.* As it is generally known, this characteristic causes that, starting from the same initial conditions (input data) the consecutive executions of the processes vary. This brings about difficulties to reproduce errors and detect them.

1.4   A single environment

Bearing in mind the success of the VDV as a method of introducing students to programming, the idea to use it to introduce concurrent programming came natural. The advantages are obvious: the students already know the enviroment and have experience with it, which allows them to focus on the specific matters of the topic.

At first, the idea to modify VDV and add facilities that would allow its use to solve concurrent programming issues was considered. The result of this analysis was that it should be fully reprogrammed. Some of the reasons behind this decision were:

- VDV is programmed in Delphi and only works with Windows. To produce a multiplatform version was a priority due to the fact that many students use free software and would not be able to install it.
- To create a full version would allow the gaining of experience and the improvement of some details (better messages in case of error, more coherent language, replace the enunciation of some primitives that can become ambiguous in other contexts, etc.)
- Finally, to structure VDV from its roots so that it can accept the extensions that allows its use to introduce students to concurrent programming.

1.5   Other aspects to consider

Even if experience shows that the tools which allow the visualization of the algorithms' execution are of great help when starting programming, it is important that the teacher pays special attention to the design of the activities that will be presented to the students.

The advantages of this tool stem from the fact that it makes code debugging easier. When the errors are visible, the students become less dependent on their teacher. Nevertheless, this can lead to the problems being solved by trial and error. Without taking anything away from this method, truth is that students need to develop other strategies to solve problems.

2   Characteristics of the Concurrent DV

En primer lugar cabe señalar que la versión de DaVinci First of all, it is important to highlight that the Concurrent DaVinci version, in addition to all the elements it has to make the teaching of concurrent programming easier and that will be briefly described below, adds a group of extensions to the Visual DaVinci, the most important being:

- Multiplatform implementation
- Improvement of the error messages when compiling and executing
- Incorporation of the string data type
- Reading variables when the execution is taking place
- Incorporation of a group of primitives (random numbers, convert text to numbers and numbers to text, etc.)
- Replace the indentation for the keywords *comenzar* and *fin* to limit code blocks
- Admission of a distribution of flowers, papers and obstacles established by the student, and the possibility to preserving it for future executions

  The main extensions for concurrency are:

- Incorporation of the abstract data type *semaphor*, both general and binary. *Semaphores* solve the typical issues of concurrency. This data type only works with the primitives *iniciarSemaforo, esperar* and *avisar*
- Incorporation of the most common short-term schedulers (FIFO, Round-robin, random). Although a concurrent program should work properly regardless the type of scheduler the operating system uses, the possibility to choose among different schedulers allows a practical observation of how they influence the execution.
- Incorporation of the possibility to manipulate the logical execution sequence. Concurrency introduces the problem of non-determinism. That is why it is essential to have a mechanism that allows both the exact reproduction of a concurrent execution that is finished (correctly or incorrectly) as the ability to execute traces established by the user ("forced traces") that lead to error situations and that, regarding non determinism, can fail to happen even when a high number of program executions is made.

3   Example

Concurrent programming is usually taught through a group of traditional examples, each of which work as a metaphor for real situations that come up in the operating systems enviroment (original cradle of concurrent programming), and that then can extend to other domains.

These examples help to set out the typical issues that come up when accessing to shared resources and when synchronizing the different processes that help to the solution of the problem is needed. One of these examples is the Bounded Buffer Problem.

3.1   The Bounded Buffer Problem

This problem is first considered taking into account a couple of processes that share a repository of predefined size. They are usually identified as *producer* (the process that adds elements to the repository) and *consumer* (the process that takes elements from the repository). Later, the problem can be generalized into ***n producers*** and ***m consumers.***

The problem presents requirements from the point of view of accessing to the use of the shared resource (the repository), and also regarding the synchronization between processes: a producer cannot add elements to a full repository and a consumer cannot take elements from an empty repository.

In DVC, the simplified problem has been represented using two robots: one is the producer and the other one, the consumer. The repository is an area of the city (Slots for Consumer in Fig. 1). The elements of the repository are flowers, which the producer picks up from other area of the city (Producer Resources Quadrant in Fig. 1), where the flowers have been placed before the execution. For the process to be executed indefinitely, once the consumer takes an element of the repository, he must place it back in the quadrant from which the producer takes it. The code of the example is the following:

```
programa ProductorConsumidor

variables //globales
  lleno:semaforoGeneral
  vacio:semaforoGeneral
  turno:semaforoBinario
  turnoCuadrante:semaforoBinario
  avenidaProd:numero
  avenidaCons:numero

  //Constantes
  AVFINAL : numero
        CALLECONS : numero
```

```
  CALLEPROD : numero

subprogramas

  procedimiento tomarFlorDeCuadrante(en calle:numero;
en avFin:numero)
  comenzar
    Pos(1,calle)
    mientras ! HayFlorEnLaEsquina
      Pos((posAv % avFin)+1,posCa)
    tomarFlor
  fin

hilos
  hilo productor (en caPro:numero;en caCon:numero;en
avFin:numero)
  comenzar
    iniciar
    mientras v
      comenzar
      esperar(turnoCuadrante)
        tomarFlorDeCuadrante(caPro, avFin)
      avisar(turnoCuadrante)
      esperar(vacio)
        esperar(turno)
          Pos(avenidaProd,caCon)
          depositarFlor
          avenidaProd := (avenidaProd % avFin)+ 1
        avisar(turno)
      avisar(lleno)
      fin
  fin

  hilo consumidor (en caPro:numero;en caCon:numero;en
avFin:numero)
  comenzar
    iniciar
    mientras v
      comenzar
      esperar(lleno)
        esperar(turno)
          Pos(avenidaCons,caCon)
          tomarFlor
          avenidaCons := (avenidaCons % avFin)+ 1
        avisar(turno)
```

```
      avisar(vacio)
      Pos(aleatorio(avFin)+1,caPro)
      depositarFlor
      fin
  fin

comenzar
  //constantes
  AVFINAL := 10
  CALLECONS := 2
  CALLEPROD := 8

  //recursos compartidos
  avenidaprod := 1
  avenidacons := 1

  //inicilizacion semaforos
  iniciarSemaforo(lleno,0)
  iniciarSemaforo(vacio,AVFINAL)
  iniciarSemaforo(turno,1)
  iniciarSemaforo(turnoCuadrante,1)

  //arrancamos los hilos
  arrancar  productor(CALLEPROD, CALLECONS, AVFINAL)
  arrancar consumidor(CALLEPROD, CALLECONS, AVFINAL)
fin
```
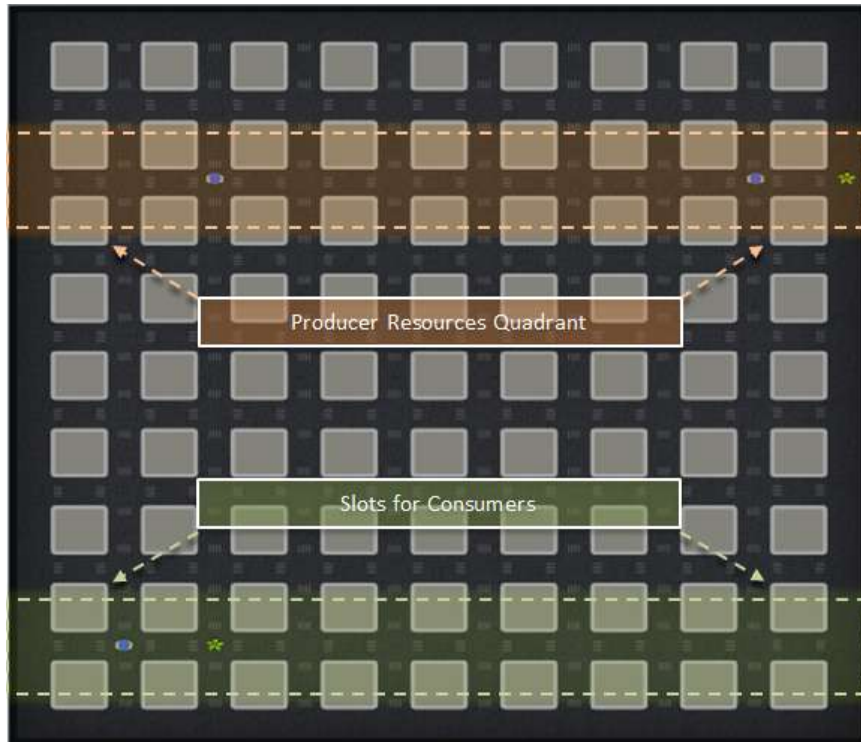
Fig. 1.

4 Future projects

The interpreter has been developed as part of a thesis. In order to show its functionality, an elementary development environment has also been created.

In the future, we are planning to work on the following aspects, among others:

- Improvement of the development environment: the plan is to change the elementary environment the current version has. The new environment will have characteristics similar to the modern environments, like NetBeans and Eclipse, and will provide useful tools for configuration, code editing and debugging and it will also improve the visualization of the city and its robots. The fact that the new environment will be similar to the currently used will be an extra advantage.
- Language extension: the extension of the specification is planned in order to study in depth different concepts related to computer programming in general. Some of these extensions are: the possibility to declare functions, the incorporation of array

data type, the improvement of the robots' characteristics (making it possible to uncoupling them from the threads of execution).

- Incorporation of other primitives and mechanisms for concurrency. It is contemplated incorporate at least messages and monitors.
- Verification of the "forced traces": currently, it is the users' responsibility to make sure that a "forced trace" is coherent with the code. In the future, the interpreter will have the ability to perform this task.

References

1. Devlin, K.: Why universities require computer science students to take math, Communications of ACM 46, 9, 37–39, (September 2003)
2. Kramer, J.: Is abstraction the key to computing?, Communications of ACM 50, 4, 36–42, (April 2007)
3. Dann, W., Cooper, S.: Education: Alice 3: concrete to abstract. Communications of ACM 52, 8, 27–29, (August 2009)
4. Utting, I., Cooper, S., Kölling, M., Maloney, J., Resnick, M.: Alice, Greenfoot, and Scratch -- A Discussion, Transactions on Computer Education 10, 4, Article 17, 11 pages, (November 2010)
5. Champredonde, R., De Giusti, A.: Design and Implementation of Visual Da Vinci. In: III Congreso Argentino en Ciencias de la Computación (CACiC 1997), La Plata (1997)
6. De Giusti, A. y otros. Algoritmos, Datos y Programas. Pearson Education, Buenos Aires (2001)
7. Feierherd, G., Depetris, B., Jerez, M.: Una evaluación sobre la incorporación temprana de algorítmica y programación en el ingreso a informática. In: VII Congreso Argentino de Ciencias de la Computación (CACiC 2001), El Calafate (2001)