A Teaching Environment for the Development of Parallel Real-Time Programs

Célio E. Morón, José R. P. Ribeiro, Nilton C. da Silva
Federal University of São Carlos - Department of Computer Science
São Carlos - SP - 13565-905 - BRAZIL

Abstract - The Teaching Environment being described here is introduced as an effective way for learning how to deal with the problems related with parallel/distributed real-time systems. The educational process has three main domains: the cognitive, the affective and the psychomotor. Normally, the student dominates the first one, but lacks more interest for the subject and a familiarisation with the challenges of real problems in the area. The environment has a visual interface with the user whose aim is to facilitate the learning process in the generation and debugging of source code of applications developed for the Parallel Kernels and in particular for Virtuoso (Virtual Single Processor Programming System - Virtuoso is a trademark of Eonic Systems - http://www.eonic.com). The environment is composed by four tools: a Parallel Programs Generator, a Worst Case Execution Time Analyser, a Scheduling Analyser, and a Parallel Debugger. The first tool is available for downloading from http://www.dc.ufscar.br/tev/tev.html and was released as Teaching Environment for Virtuoso (TEV). The second of these tools is already under implementation and the last two tools will be available in a near future. Roughly, this Visual Environment should be able to provide continuity in the development of projects using the most common development methods (traditional or object-oriented), by offering support during the phases of implementation, debugging and testing.

1. Introduction

Usually, students from the computer science or computer engineering courses learn the basic concepts of the disciplines related with parallel/distributed systems, without necessarily having acquired the effective ability of intuition, rapid reasoning and assurance in problems related with those concepts.

This is because the educational process has three domains: the cognitive, the affective and the psychomotor. Normally, the student dominates the first one, but lacks more interest for the subject and a familiarisation with the challenges of real problems in the area.

There is a general agreement in the educational literature that learning processes that result in meaningful knowledge exhibit certain properties. For example, it is often said that meaningful learning is cumulative and constructive. It is cumulative in the sense that the learner’s prior knowledge determines the new knowledge that is learned, and is constructive in the sense that the new information must be elaborated and related to previous information in order that it can be learned [1, 2].

Learning is a dynamic process where students do not receive the knowledge in a passive manner, they are active builders of what is learned. This knowledge cannot be reduced to a text, and it originates in the need to solve problems. Efficiency in solving problems is the main criteria for expansion of knowledge. Problem solving usually leads to the generation of new parts, which in turn usually contain parts of already existing knowledge.

The basic idea behind our tool is that knowledge is constructed through the interaction of the learner with the basic objects of the environment. According to some educational methods the learning process can be characterised by questions about the objects such as: “what it is made of”, “what can be done”, “what the actions lead to”. Thus, to learn the meaning of an object corresponds to the assimilation of schemes of actions of utilising or constructing objects and to learn the meaning of an action corresponds to the assimilation of schemes of actions of transforming objects.

The environment being described here offers a set of consistent resources oriented to the representation, exploration, and construction of knowledge. These resources facilitate the elaboration and enhancement of the learning situations, while at the same time, favour the more elaborated cognitive process. The resources are directed to the student and represent an important role in the knowledge construction process. The environment can be seen as a place in which learners can have individualised learning experiences, regardless of their individual differences, previous experiences, or cognitive states.

The main goal in using TEV is to individualise the learning process; stimulate and motivate the student; promote self-esteem; and finally present the curricular topics in an attractive, creative and integrated manner.

The student can have control over his learning and choose his own educational path through a consistent and intuitive interface that incorporates basic parallel real-time objects. In this way, the student is allowed to learn at his own pace, and the learning is orientated by the satisfaction of the needs and personal interests. The student’s control emphasises individual aspects, such as: creativity, discovery, invention, innovation, and so on. As final results, it is possible to achieve an increase on the level of attention and positive behaviour, while at the same time, a diminish anxiety. However, the highest achievement is the ability the student acquires in creating his own knowledge.

The remainder of this paper is organised as follows: Section 2 describes the process of developing parallel real-time programs. Section 3 shows the structures of the kernel Virtuoso represented in TEV. Section 4, Section 5, Section 6 and Section 7 are dedicated to the description of the
The emergence on the market of processors at low cost is making possible the construction of powerful parallel systems, which are able to support high-performance applications. Examples of these kinds of applications are found in multimedia, virtual reality, robot vision, medical image processing and fault-tolerance [3, 4]. However, the development of these systems, is often hard to manage, mainly due to:

1. The technology used for the construction of the old systems basically dictated the sequential style for the development of programs. Most programming languages reflect the behaviour of sequential processors.
2. Most software systems for sequential computers hide the hardware architecture from the user. However, for a system with several processors the programmer needs to be aware of the hardware architecture in order to implement the programs.
3. The implementation of traditional tools is more complex in parallel systems. As a consequence, the availability of tools for this kind of systems is scarce.
4. In parallel systems, it is very common to find difficulties at the transition between analysis, design and implementation. During implementation programmers do not have adequate tools for mapping the requirements into the source code.

The Teaching Environment being described here is introduced as an effective way for learning how to deal with the problems mentioned above. This work describes a Teaching Environment for the Development of Parallel Real-Time programs, whose aim is to facilitate the learning process involved in the generation and debugging of source code of applications developed for the Parallel Kernel Virtuoso [5]. When offering support to the kernel, a set of tools is required to help in the development of parallel applications, debugging and analysis of real-time requirements. Within these tools it is worth mentioning: 1- Parallel Programs Generator, 2- Worst Case Execution Time Analyser, 3- Scheduling Analyser, and 4- Parallel Debugger. These compose the Teaching Environment for the Development of Parallel Real-Time Programs, an integrated programming environment for the development of applications executed in a parallel machine. This Visual Environment, should be able to provide continuity in the development of projects using the most common development methods (traditional or object-oriented), by offering support during the phases of implementation, debugging and testing. Figure 1 shows the structure of TEV.

Figure 2 shows the interface of the environment with the user and a small program developed using TEV.
2.3 Scheduling Analyser (SA)

The aim of this tool is to foresee whether the requirements of the system will be reached or not. The Scheduling Analyser simulates the execution of the processes in order to evaluate whether the execution in a real-time system is viable. The analyser is also able to inform where the faults were in order to allow the programmer to fix them. Based on the estimates produced by the Worst Case Execution Time Analyser, the Scheduling Analyser produces a graphical representation of the system’s tasks. This graphical representation enables the identification of the tasks that are not going to meet the real-time requirements. The aim of the Scheduler Analyser, therefore, is to verify, before the execution of the parallel program, whether the real-time requirements will be met. In case they will not, the system can be modified either by the addition of more processors, or by modifying the source code.

2.4 Parallel Debugger (PD)

The aim of this tool is to allow the on-line debugging of the programs developed in the Visual Environment. The Parallel Debugger allows the programmer to debug the application in the same graphical environment used to develop it. The graphical notations produced by the Parallel Program Generator and the Scheduler Analyser are used as a visual interface for the data shown to the user during the debugging.
process. A more detailed graphical description about the internal execution of each task (variables, execution flow, calls to the kernel's services) is shown by using the diagram produced by the Scheduler analyser.

3. Structures of the Kernel Virtuoso in TEV
The graphical notation used by the Parallel Programs Generator tries to integrate the three basic components (functional, behavioural and structural) of a parallel real-time program into a single graphical representation. As in most visual environments, this representation is based on graphs composed of nodes (tasks) and arcs (message flow). However the graph is extended in order to represent also the data structures that control the communication and synchronisation between the processes (semaphores, mailboxes, resources, etc.).

Although the graphical representation adopted by the Parallel Programs Generator is a generic one, its semantic is based on the programming model established by the kernel Virtuoso. The next section presents the basic characteristics of this model.

Virtuoso offers a multilevel programming system built around the optimised nanokernel. The ISR (Interrupt Service Routine) levels are used to directly control the hardware interruptions of the processor.

Level 3 (nanokernel) is composed of several tasks of reduced context, called processes. Each process is a routine written in assembly language that may call C functions. Level 4 is composed of more than 70 services of the microkernel, which can be called from the C language. The basic components of this level are the microkernel's objects. The main objects of the microkernel are tasks, semaphores, queues, mailboxes, memory partitions, resources and timers.

Class Task - A task is a program module which exists to perform a defined function or a set of functions. A task is independent of other tasks but may establish relationships with other tasks. These relationships may exist in the form of data structures, input/output, or other constructs. Virtuoso uses the multitasking concept, which appears to give the processor the apparent ability to be performing multiple operations concurrently. As a sequential machine, a processor cannot be doing two or more things at the same time. However, with the functions of the system segregated into different tasks, the effect of concurrency can be achieved. Efficient use of the processor can be obtained by using the time a task might wait for an event to occur to run another task. Figure 2 shows a program with two tasks.

Class Semaphore - Semaphores are used to synchronise/handshake between two tasks and/or events. A signalling task will signal a semaphore while there will be another task waiting on that semaphore to be signalled. One can wait on a semaphore with a time-out or return from the wait if no semaphore is signalled. This can be useful to make sure that the task does not get blocked. Figure 2 shows two tasks and a semaphore in a program.

Class Message - Messages are used between a sender and a receiver task. This is done using a mailbox. The mailbox is used as an intermediate agent that accepts messages. Messages work with arbitrary sizes and permit a selective transport between sender and receiver, including the specification of the priority of the message. Figure 3 shows a program using a Mailbox in TEV.

Class Timer - This class of calls permits an application task to use a timer as part of its function by allocating a Timer object. From then on, the timer can be started to generate a timed event at a specified moment (one shot) or interval (cyclic). This event can then signal the associated semaphore. Timers are mainly used to regulate the execution of tasks with respect to a required timely behaviour. Figure 4 shows a program using a timer

Class Resource - The resource protection calls are needed to assure that access to resources is done in an atomic way. Unless the processor provides real physical protection, the locking and unlocking of a resource is in fact a convention that all tasks using a resource must follow.

Class Queue - Queues are also used to transfer data from any task to any other task but here the data is actually transferred in a buffered and time ordered way. The advantage is that no further synchronisation is required between the enqueuer and the dequeuing task, permitting the enqueuer to continue. Another advantage of this mechanism is that a queue also acts as a “port” (for example to access the console from any node in the system).

Class Memory - In any system, memory is a resource for which tasks are competing. Memory management is an area where various techniques can be applied. Many techniques are very fast and use elegant models for allocation and deallocation of memory.

4. Graphic Editor
The graphic editor offers a user-friendly interface for the development of Virtuoso applications. This editor is used to create the data structure of the application and generate part of the source code.

Using the graphic editor, the user creates a diagram that represents the most important characteristics of the application. In this diagram, a different symbol is used for the representation of each kind of object (task, semaphore, resource, mailbox, etc.), and the calls to the microkernel appear as links connecting these objects. The objects represented by the graphical editor can be divided into two categories: configurable and executable objects.

The graphical symbol of each object is delimited by a rectangle, within which the object's name and a figure, representing the kind of object (semaphore, task, etc.), are drawn.

In the graphic editor, relationships between objects are represented through connections. These can be defined as visual representations of the calls that the executable objects (tasks and functions) make to the services of the microkernel. A connection is represented in the graphic editor through two components: a connection line and a graphical symbol. A connection line is used to interconnect the object that called the service (a task or a function) to the object under which this service was executed. The graphical symbol indicates the microkernel primitive associated with this
A connection is placed, by the user, on one of the line's segments interconnecting the involved objects. The kind of connection, that is, the microkernel service that it represents can be easily distinguished by the format and colour of the graphical symbol. Figure 2 illustrates the connections between two tasks using TEV for Virtuoso.

![Figure 2 - Connections Example](image)

5. Text Editor
The graphic editor allows a representation, at the high level, of the data structures and calls to the microkernel services. However, the executable objects’ source code does not contain only calls to the microkernel. The functionality of these objects is determined mainly by segments of the code written by the user. These segments vary according to the application, and are therefore better defined through a textual form rather than by graphical representations.

The text editor provides the user with the facilities to write, in C language, the code of the executable objects created by the graphic editor. The text editor is a component of the Parallel Programs Generator and offers, in addition to the basic facilities found in common editors, other resources to integrate the textual part of the application (source code of the executable objects) with the graphical part (executable objects and connections).

When a new application is initiated, the basic skeletons of the special files are automatically generated by the text editor. This skeletons can be updated later with additional information.

6. File Manager
The user’s application may be stored in a project’s file. This file contains all the information necessary to restore the graphical and textual parts of the application.

The module responsible for storing and restoring the project's files is the file manager. During the saving process, the file manager reads the internal data structure stored in the main memory, makes some conversions, and then stores this data in the project's file. When the application needs to be restored, the file manager carries out the inverse operation.

7. Code Generator
The code generator is responsible for producing the final output of the Parallel Programs Generator, that is, the files necessary to compile and linkedit the user's application. Two kinds of files are created by the code generator: files containing the application's C code and a makefile file.

The code generator produces source files, in C language, for all the microkernel objects being graphically represented. Two kinds of source code are produced. The first ones
contain the code that implements the microkernel objects' data structure. The second ones are designed as header files (.h files), that can be inserted within the application's source code.

For each executable object (task or function), a third file is also generated. This file contains the source code executed by the object.

Also, the code generator creates makefile files, which can be invoked by the make utility in order to produce executable files for the application. The make and run commands from the main menu can be used to compile, linkedit and execute the application. In this way, the user does not need to leave the visual environment to execute these commands.

8. Conclusions
A significant limitation for the teaching of parallel real-time systems is the lack of adequate programming tools, mainly for supporting the final stages of the development life cycle. This work presented a development environment, called TEV, that supports the design and implementation of parallel real-time applications executed with the support of the parallel kernel Virtuoso.

The graphical notation used tries to integrate the three basic components of a parallel real-time system (behavioural, structural and functional), into a single graphical representation, facilitating therefore the understanding of the system as a whole.

A prototype of the Visual Environment for the 3.1 version of the Parallel Kernel Virtuoso has already been developed and is available for downloading, where some examples, as well as a case study that demonstrates how the tool works, can be found. This environment will be used as a teaching environment for the users of the kernel Virtuoso.

The environment has been successfully used as a tool to teach traditional operating systems. The goal of using the Environment as an individual learning method was achieved with full success. As a result we worked with more motivated and stimulated students, increasing dramatically their self-confidence and self-esteem. In short, the course was more attractive, creative integrating theory and practice in an efficient manner. As a final remark, it was possible to achieve an increase on the level of attention and more positive behaviour, while at the same time, a decrease on anxiety.

References