A Visual Environment Integrating Design, Implementation and Debugging in Parallel Real-Time Systems

Célio E. Morón[1], José R. P. Ribeiro [1], José Hiroki Saito [1], Hanumant S. Sawant[2], Reinaldo R. Rosa [2]

[1] Federal University of São Carlos - Department of Computer Science
São Carlos - SP - 13565-905 - Brazil
{celio@dc.ufscar.br, ribeiro@dc.ufscar.br, saito@dc.ufscar.br}

[2] Astrophysics Division (DAS)
National Institute for Space Research-INPE, Brazil
{ sawant@das2.inpe.br, reinaldo@lac.inpe.br}

Abstract—
Due to the culture of sequential programming, the lack of tools and the inherent difficulties of parallel programming, most programmers find hard to design and evaluate parallel real-time programs. As a result, a major problem found in the development of parallel real-time systems is the difficulty to produce rapid prototypes of the application, and quite often, the development of this kind of systems is behind schedule. This work is part of a bigger project financed by FINEP, with the title "Arquitetura Paralela Usando DSPs" (Parallel Architecture Using DSPs), whose aim is to use a parallel system for processing and visualising tomographic images from the solar atmosphere in real time. In this way, it can be possible to have the 3D reconstruction of the solar corona, from the angular tomography, as well as from the spectral one. This paper presents a Visual Environment which 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. The environment has a visual interface and the main aim is to facilitate the development process and debugging of source code of applications developed for parallel kernels, 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 version of the tool is available for download from http://www.dc.ufscar.br/~tev and was released as Teaching Environment for Virtuoso (TEV).

Keywords— **Visual Environment, High Performance, Real Time, RTOS.**

I. Introduction

The main goal of the project is the construction of a parallel system for high performance processing using the Elebra's PAD called VORTIX, working together with a set of DSPs (Digital Signal Processing). The project consists of constructing a parallel system with the capacity to support realistic applications, involving a reasonable amount of parallel processing. The development of this project is a joint effort between the Department of Computer Science at UFSCar and the Astrophysics Division at INPE, and is aimed at using the parallel system for carrying out the processing and visualisation of tomographic images from the solar atmosphere in real time. In this way, we could have the 3D reconstruction of the solar corona, from the angular tomography, as well as from the spectral one. This parallel system will be accessed remotely through high-speed networks, and will also be able to attend other kinds of applications, such as: multimedia, virtual reality, scientific visualisations, image processing for medical applications, weather forecasting, industrial automation, robot vision, and so on.

In the VORTIX side we will have the third-party software, whereas in the DSPs we will have a Visual Environment working together with the kernel Virtuoso. The kernel requires a set of tools to help on the development, debugging, and analysis of real-time requirements. These tools need to be integrated in a visual environment in order to provide a user-friendly and self-explaining interface. In addition, this visual environment should be able to give continuity to the development of projects that use the most common development methods (traditional and object-oriented). Within this set of tools we can mention the *Parallel Programs Generator, a Debugger, Worst-Case-Execution-Time Analyser, and Scheduling Analyser*. Through these tools, the user can create a visual representation of the application's structures (tasks, semaphores, timers, mailboxes, queues, and so on) and provide the characteristics of each structure. From this visual representation, the tool allows the automatic generation of the application's C code.

Solar Atmosphere Dynamic Images obtained with high sensitivity and high spatial and temporal resolution, have allowed to identify the solar atmosphere as a highly heterogeneous environment. X-rays films of the coronal dynamics, observed by sensors on board of the Yohkih satellite, have shown that the structures in X-rays are not in a static equilibrium, but rather in a slow and continuous evolution [ALL 92, TSU 93, UCH 97].

Furthermore, temporal-spatial fluctuations of soft X-rays intensity have shown evidence of the existence of several kinds of transient phenomena associated to the plasma-magnetic structures. The study of spatial-temporal fragmentation of the energy in X-rays have allowed, for the first time, to characterise turbulence located in the solar corona plasma.

However, physical and geometrical parameters associated with this approach are subject to systematic errors due to the spatial limitation imposed by bi-dimensional images. Actually, the basic topology in that solar atmosphere in X-rays consists of loops, where each element is generally described by a tri-dimensional model, such as the observed discrepancies found in most models and observations is due to the fact that in the models the main physical parameters (temperature, density and magnetic field) and geometric (height, rotation and position) can vary as a function of the three spherical co-ordinates. Motivated by this limitation, efforts have been made in order to allow the real-time tri-dimensional visualisation of the solar atmosphere.

The most common technique for the reconstruction of 3D images from 2D ones is an angular tomography; this technique consists of recording *n* 2D images of a 3D object observed from different angles. The particular case where *n* = 2, is called stereoscopy. In the solar case, the angular tomography can be obtained in two ways: by using *n* satellites positioned at different latitudes; and by using a single satellite that considers an angular variation resulting from the solar rotation itself.

However, the second method is restricted to the observation of structures that are stable in time.

A more recent technique for the reconstruction of structures in the solar atmosphere, is the spectral tomography, which instead of using the angular variation, uses the frequency variation, which is a typical phenomenon in plasmas that present a sensitive gradient in density [GRE 95]. Thus, the spectral tomography demands simultaneous observation, of the same event, in different frequencies. Considering the existing radio-interfometers already in operation, the advantage of this method is that almost all the observational apparatus required is already available. Co-ordinated observations for obtaining images in X-rays and UV could be incorporated in order to allow a more accurate estimate of the temperature and gradient density of the solar corona [ASC 95].

This paper focuses on the visual Environment used to develop applications on this parallel system, mainly on the DSP side of the system.

The remainder of this paper is organised as follows: section 2 describes the Visual Environment; section 3 describes some related work; section 4 presents the tools that compose the Visual Environment; section 5 shows the structures of the kernel Virtuoso represented in TEV. Finally, section 6 presents the conclusions about this work.

II. THE VISUAL ENVIRONMENT.

The technological evolution has made possible to buy processors at low cost, making viable the construction of parallel systems, able to support high performance applications. Examples of this kind of applications can be found in multimedia, virtual reality, robot vision, medical image processing, and fault-tolerant and hard real-time systems [MOR 96, MOR96b]. With the massive propagation of the World Wide Web, a new range of parallel applications have emerged. This has made possible the remote access of parallel machines dedicated to specific applications.

Typically, a parallel application is composed by a set of individual modules which, when combined, represent the application's natural parallelism. This natural parallelism can be divided into two parts: parallelism of data and parallelism of control. According to the application, the composition of both parallelisms differs on its proportion. In general, the use of parallelism is applied in applications where the parallelism of data is at least three times greater than the parallelism of control.

Traditionally, software developed for parallel computation has been aimed at maximising the program's performance rather than the programmer's productivity. However, to increase productivity and, at the same time, achieve a more generalised use of parallel systems, the software has to work in an heterogeneous environment; that is, it is necessary to achieve an integration with the technology available for sequential systems. In addition, it is necessary that the software produced be portable and able to execute in traditional systems, without major modifications, although with a lower performance.

The development of parallel real-time systems, is often hard to manage, mainly due to: 1) the technology used for the construction of the old centralised systems basically dictated the sequential style for the development of programs, as a consequence most programming languages reflect the behaviour of sequential programming; 2) lack of portability; 3) the implementation of traditional tools is more complex for parallel systems; 4) it is common to find difficulties at the transition between analysis, design and implementation; 5) during implementation programmers do not have adequate tools for mapping the analysis and design requirements into the source code; and 6) the difficulty to produce rapid prototypes of the application usually results in development runnning behind schedule.

The Visual Environment for the development of Parallel Real-Time Programs offers an alternative for solving the problems above. This environment is composed by a set of tools that support the development of parallel real-time programs, from the generation of code to the phases of debugging and tests. These tools are integrated through a common graphical interface, that offers a high level environment for the construction of parallel real-time applications. In this way, the Visual Environment unifies the phases of design with those of implementation, visualisation, debugging, and validation of parallel real-time programs.

III. RELATED WORK

This tool is aimed at supporting the generation of source code that will execute in the parallel machine. Through this tool, the programs can be constructed based on a graphical model of the application, which is used to integrate the other tools of the Visual Environment.

For the development of the Parallel Programs Generator, the first step was the definition of the tool's architecture. This definition was based on a survey carried out to study the existing graphical environments for the development of parallel real-time applications. Although there are many graphical environments available for the development of parallel systems, there is no agreement as to the services and graphical representations that these environments should offer.

Some visual environments try to visually describe the parallel program through an extended concurrency map [CAI 95]. This approach offers the advantage of allowing a hierarchical representation of the parallel program, and is quite adequate to represent fine grain parallelism. However, concurrency maps do not offer an adequate notation for representing data structures at the high-level (such as semaphores, resources, mailboxes, etc.), that are present in most parallel programming languages. Most visual environments use graphs to represent parallel applications. Graphs are useful to describe processes structures, data dependency, performance visualisation, and so on.

Examples of visual environments based in the use of graphs are: TRAPPER [SCH 93], GRADE [DOZ 96], Millipede [ASP 91] and PVMGraph [JUS 96]. What differentiates these environments is the abstraction level and the contents of the information being graphically represented.

The TRAPPER environment describes the applications at two abstraction levels: whereas the parallel structure of the program is described through a processes graph, the sequential components are represented in a textual form. The graph is formed by nodes and arcs, where nodes represent the processes and arcs the communication channels. Each process being represented has a set of input and output ports, from where the arcs representing the communication channels start. The visual association between the ports and the processes, as well as the use of two abstraction levels (graphical and textual) are characteristics also found in the Millipede and PVMGraph environments.

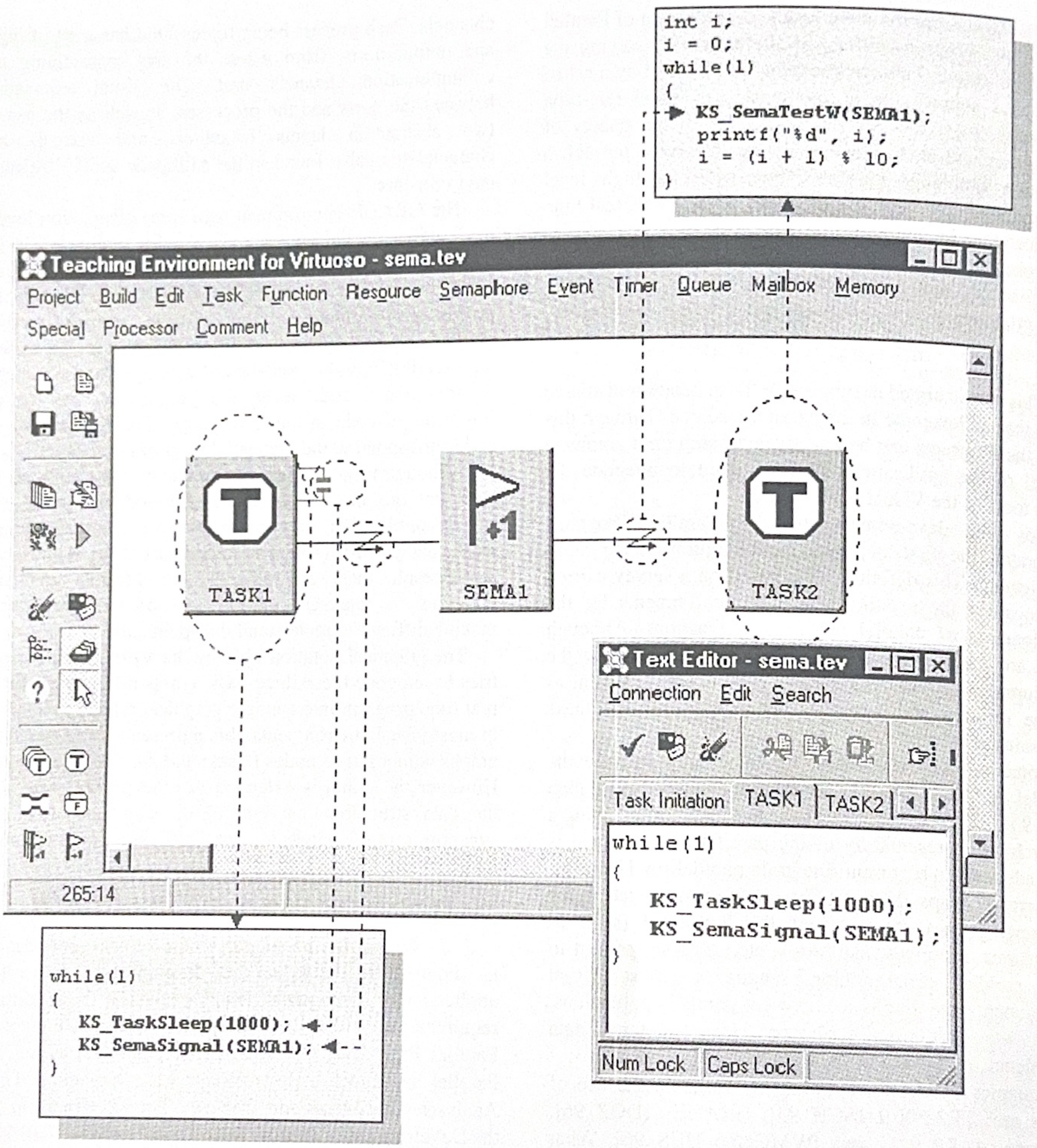
The GRADE environment uses three abstraction levels to describe parallel applications. At the application level, the processes, groups of processes and communication ports are described graphically, but the functionality of the processes is omitted. The process level describes graphically the control flow between the processes, representing only the exchange of messages between them. At the textual code level, the programmer can define fragments of code in the C language. These fragments of code correspond to the sequential parts of the application.

Although most of these environments offer facilities to represent the functional and behavioural aspects of the parallel application, they are unable to explicitly associate these characteristics to the structural part of the application. As a result, these environments use different graphical notations to represent each aspect of the application, making difficult to understand the application as a whole.

The graphical notation used by the Visual Environment tries to integrate these three basic components 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.).

IV. TOOLS WITHIN THE ENVIRONMENT

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) Scheduling Analyser, 3) Parallel Debugger and 4) Worst Case Execution Time Analyser. These tools compose the Visual Environment for the Development of Parallel Real-Time Programs [RIB 98], 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 main interface of TEV.


Teaching Environment for Virtuoso - sema.tev
Project Build Edit Task Function Resource Semaphore Event Timer Queue Mailbox Memory
Special Processor Comment Help
TASK1
SEMA1
TASK2
265:14
Text Editor - sema.tev
Connection Edit Search
Task Initiation TASK1 TASK2
Num Lock Caps Lock

```
int i;
i = 0;
while(1)
{
  KS_SemaTestW(SEMA1);
  printf("%d", i);
  i = (i + 1) % 10;
}
```

```
while(1)
{
   KS_TaskSleep(1000);
   KS_SemaSignal(SEMA1);
}
```

```
while(1)
{
   KS_TaskSleep(1000);
   KS_SemaSignal(SEMA1);
}
```

Fig.1 TEV's Graphical and Text Editor

A. *Parallel Programs Generator.*

This tool is aimed at supporting the generation of source code for the programs executed in a parallel machine. In the Parallel Programs Generator, applications are built through the construction of a graphical model, which is used to integrate the other tools from the visual environment. This model is represented by a graph, where nodes denote the data structures that compose a parallel program (tasks, signals, resources, mailboxes, etc.), and arrows denote the communication operations and synchronisation between the structures. The information in the graphical model can be complemented with textual descriptions, that is, segments of code written by the user. The application's graphical and textual information is stored in the project's file. Based on this information, the Parallel Program Generator automatically generates the source code of the application and the *makefiles*, which are used for the compilation and linkage of the source code produced.

B. Scheduling Analyser.

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. If not, the system can be modified either by the addition of more processors, or by modifying the source code.

C. Parallel Debugger.

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.

D. Worst Case Execution Time Analyser.

This tool is aimed at calculating the actual time of execution of the Worst Case Execution Time [PUS 89] of the tasks executed by the kernel. In order to calculate this time, the tool carries out a lexical and syntactical analysis of the source code produced by the Parallel Programs Generator, constructs an abstract syntax tree of the application and then traverses through the tree to find the path with the worst case execution time. This calculation is carried out based on a table that contains all possible instructions generated by the compiler.

The use of the WCET is necessary to guarantee that the time requirements of a real-time system are met. The WCET is calculated using the path with the worst case of time. This path is that one, among all possible paths of the task's execution flow, that will take more time to be executed. To calculate the WCET, the Worst Case Execution Time Analyser (WCETA), carries out an analysis of the tasks' source code, which is informed by the Parallel Programs Generator (PPG).

The WCETA allows a visualisation of the tasks' data structures through a graphical representation, where it is shown the main commands of the C language, the kernel primitives and the corresponding WCETs. This shows some of the advantages offered by the WCETA if compared to traditional tools: it facilitates the identification of the points where the task's overhead occurs; since the task's graphical representation is optimised by showing only those commands the programmer is interested in, the tool facilitates the understanding of the task; the possibility of using the graphical representation in the debugging process; it facilitates the allocation of tasks to processors.

V. 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[VIR 99] 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 classes such as: 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 gives the processor the apparent ability to perform multiple operations concurrently. In a sequential machine, a processor cannot do 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.

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.

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 message's priority.

Class Timer - This class of calls allows 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.

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" (e.g., to access the console from any node in the system).

Class Memory - In any system, memory is a resource for which tasks compete. 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.

E. Graphical Editor.

The graphical 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 graphical 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 graphical 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) do to the services of the microkernel. A connection is represented in the graphical 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 connection. This symbol 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.

F. Text Editor.

The graphical editor allows a high-level representation 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.

G. File Manager.

The user 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.

H. 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 the executable objects (tasks and functions), a third file is also generated. This file contains the source code executed by these objects.

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.

VI. Conclusions

A significant limitation for the development of parallel real-time systems is the lack of adequate programming tools, mainly those that support the final cycles of the life cycle. This work presented the Visual Environment for the Development of Parallel Real-Time Programs, whose main aim is to facilitate the implementation, debugging and testing of parallel real-time applications.

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

A prototype of the Visual Environment for the 4.1 version of the Parallel Kernel Virtuoso has already been developed and is available for downloading, where some examples can be found, as well as a case study, that demonstrates how the tool works.

This environment was used to generate the code of an application using image processing [MUC 98] consisting of images created by Brownian motion, where measures were taken by fractal dimension. The generation of code was obtained approximately in an hour, showing therefore the potential of the tool in producing rapid prototypes.

Acknowledgments

We thank the financial support from FINEP (Financiadora de Estudos e Projetos) and FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo).

References

[ALL 92] ALLISTER, A et al. - PASJ, 44, L205, 1992.

[TSU 93] TSUNETA, S. e Lemen, J. R. - Physics of Solar and Stellar Coronae, pp 113, 1993.

[UCH 97] UCHIDA, Y. - Physics of Solar and Stellar Coronae, pp 97, 1993.

[GRE 95] GREBINSKY, A S.; OPEIKINA, L. V. e BOGOD, V. M. - Lecture Notes of Physics, May 1995.CHIDA, Y. - Physics of Solar and Stellar Coronae, pp 97, 1993.

[ASC 95] ASCHWANDEN, M. J. Lecture Notes of Physics, May 1995.

[MOR 96] MORON, C., "Designing Adaptable Real-Time Fault-Tolerant Parallel Systems", *10th Int. Parallel Processing Symposium - IPPS*, April 1996, Honolulu, Hawaii.

[MOR96b] MORON, C., "Designing a Real-Time Recoverable Action", *3rd International Workshop on Real-Time Computing Systems and Applications (IEEE)*, October 1996, Seoul, Korea.

[DOZ 96] DÓZSA, G.; KACSUK P.; FADGYAS T. , "Development of Graphical Parallel Programs in PVM Environments", In Proc. of 1st Austrian-Hungarian Workshop on Distributed and Parallel Systems, Miskolc, Hungria, October 1996, pp. 33-40.

[ASP 91] ASPNÄS, M., BACK R.J.R., LÅNGBACKA T., "Millipede - A Programming Environment Providing Visual Support for Parallel Programming", Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, N° 129, 1991.

[JUS 96] JUSTO, G. R. R., "PVMGraph : A Graphical Editor for the Design of PVM Programs", Technical Report, University of Westminster, May 1996.

[PUS 89] PUSCHNER, P. e KOZA C., "Calculating the Maximum Execution Time of Real-Time Programs" The Journal of Real-Time systems, pg. 159-176, September 1989.

[MUC 98] MUCHERONI, M. L.; MORON, C. E., MALARA, R. and RIBEIRO, J. R., "Environment and Application in Parallel Machine ArchMDSP", EUROMICRO Workshop on Network Computing, September 1998.

[VIR 99] VIRTUOSO - The Virtual Single Processor Programming System", *User Manual, Version 4.1, EONIC SYSTEMS*

[RIB 98] RIBEIRO, J.R.P., SILVA, N.C. da, and MORON, C.E., *A Visual Environment for the Development of Parallel Real-Time Programs,* in LNCS (Lecture Notes in Computer Science) - Springer Verlag volume of the IPPS Workshops Proceedings (IEEE), Orlando, FL, USA, March 30 to April 3rd, 1998.

[CAI 95] CAI, W.; PIAN T.L., TURNER S. J. "A Framework for Visual Parallel Programming", In Proceedings of Aizu International Symposium on Parallel Algorithms/Architecture Synthesis, IEEE Computer Society Press, Japan, March 1995.

[SCH 93] SCHAEFERS, L.; SCHEIDLER, C.; KRAEMER-FUHRMANN, O., "TRAPPER - A Graphical Programming Environment for Industrial High-Performance Applications", Parle (Parallel and Languages Europe), Muenchen, June 1993, pp. 11.