A Distributed Debugging Tool for a Parallel Software Engineering Environment*

José C. Cunha, João Lourenço, Tiago Antão
Universidade Nova de Lisboa
Departamento de Informática
2825 Monte de Caparica
Portugal
{jcc,jml,tra}@di.fct.unl.pt

October, 1996

Abstract

We discuss issues in the design and implementation of a flexible debugging tool and its integration into a parallel software engineering environment.

1 Introduction

We discuss the design and implementation of a debugging tool (DDBG) that we have been developing as part of a software engineering environment for parallel applications in the scope of ongoing SEPP and HPCTI projects of the COPERNICUS Programme [2] [13]. The main goal of the SEPP environment was to develop a suitable set of tools and to focus on their integration into a flexible and user-friendly environment. Namely, the abstractions that are being offered to the user by each tool in the environment are all related to the high-level graphical parallel programming model that is supported by the GRAPNEL system. The GRAPNEL model defines a graph-based parallel programming language that is described elsewhere [5] [12], supporting a structured style for designing parallel applications. It has a companion set of integrated tools, such as a graphical editor, a compiler to an intermediate textual representation, and to the C language extended with PVM primitives, a visualization tool, a simulator, a testing and debugging tool, and monitoring and a load-balancing tools. The development of those tools is a major effort being undertaken within the mentioned projects [13][4][3].

Here we discuss design and implementation issues for one component of such parallel programming environment: the debugging tool DDBG.

Although there is currently a large diversity of debugging tools, there is still a need to provide better functionalities offering the user more help in the identification and examination of the causes of incorrect program behavior,as well as more adequate integration with other tools of the parallel development environment (such as the graphical editor, the programming system, the simulation and visualization tools). Additionally some of the existing systems are very much dependent upon a specific hardware or operating system platform.

The SEPP and HPCTI projects have provide an excelent environment to test the design issues involved in improving debugging tools. The following initial requirements have played a major role in the design and implementation of the debugging tool:

- Close integration of testing and debugging tools
- High-level debugging at the graphical editing level
- Architecture independence

* In "*Proceedings of EPTM'96, 1st European Parallel Tools Meeting*", ONERA, France, October 1996.

Concerning the first requirement, the development of a methodology and tool to aid the user in the process of idebtifying paths which should be generated and tested, is a key component of an advanced testing and debugging environment [2][6]. However, an important aspect of the approach followed in our project is to allow the testing and evaluation stages to be performed through a close interaction with the dynamic debugging tool. This is achieved by supporting user controlled execution of the paths under test, allowing the user to inspect program behavior at the desired level of abstraction and with the guarantee of the reproducibility of its execution. During the testing stage, distinct testing scenarios are identified, their corresponding program paths are generated, and then corresponding scripts are produced so that it is possible to submit them to a real parallel execution. Based on that information, generated during the testing phase, it is possible to control the replay of program execution, suitable instrumented with calls to the debugging system breakpoints, stepwise execution, etc. The detailed description of the functionalities of such testing tool, called STEPS, that is being developed by our partners of the Technical University of Gdansk, is found in [6]. Its interfacing to the DDBG distributed debugging system is directly supported by the mentioned script files which consist of sequences of debugging commands such that specific execution paths are followed under real execution.

Concerning the second requirement, the GRAPNEL editor offers an user-friendly interface that allows to invoke debugging commands with reference to the graphical entities that are displayed in the user windows. On the other hand, there is a requirement to display in a convenient way the debugging outputs such that only GRAPNEL abstractions should be handled by the user at this level. This offers s a very high-level interface to the user, such that the information on specific debugging commands is directly related to the GRAPNEL source program, e.g. by highlighting corresponding entities in the graphical representation, and their corresponding lines of source code in the textual program representation. The detailed description of the functionalities of such testing tool, called GRAPNEL, that is being developed by our partners of the KFKI-MSZKI, the Research Institute for Measurement and Computing Techniques of the Hungarian Academy of Sciences, and the University of Miskolc, is found in [12].

Concerning the third requirement, the SEPP project addresses target parallel architectures that are in constant evolution, so a need has arisen to provide a reasonable degree of architecture independence, as well as the ability to adapt to a large diversity of parallel platforms and runtime systems.

2 The DDBG Debugging Tools and Topics for Discussion

The evolution of parallel and distributed systems towards more user-friendly environments requires a very flexible software development platform for the experimentation with new programming models and the corresponding development tools, e.g. for monitoring, debugging, animation, visualization, and performance analysis. After several years of experimentation with these parallel computing platforms, the current state of the art clearly shows a need to provide a more unified framework to support the implementation of high-level debugging functionalities. This framework must address two fundamental issues:

- A well-defined interface must be provided to be used by high-level tools of the parallel development environment, namely graphical editors, graphical interfaces, runtime support systems for distinct parallel and distributed language models, and testing and high-level debugging tools;
- A well-defined interface must be provided to the underlying operating system and hardware platform, assuring portability and adaptability of the debugging support architecture, while still allowing efficient implementation on top of each specific physical environment.

Each of the above issues may be separately addressed, to a certain extent.

Concerning the first issue, we have defined an interface to a library of debugging primitives, and we have implemented it on top of PVM [10]. From the user point of view, any application or tool can be linked with the interface library and access all the distributed debugging functionalities. The debugging functionalities may be summarized as follows:

- Dynamic attachment and deattachment of debugger instances to already running distributed processes;
- Control of remote debugger instances from a central debugging user interface;
- An interface library that gives access to such control of remote debuggers, and which can be used by high-level tools, like a graphical editor, and testing tools;

- An event trace is collected with minimal information to support program replay in PVM programs. This allows reproducible behavior and will make the debugging control commands available during a replay session;
- A checkpoint facility under replay mode will support execution replay from an intermediate point, instead of from the beginning of the program only.

Currently there is a working prototype implementing the first three of the above functionalities. Full support for program replay and checkpointing is currently under development.

Concerning the second issue, the debugging tool has a distributed organization consisting of multiple monitor/debugger instances which are scattered on the nodes of a PVM platform. The prototype runs on the PVM environment, and relies upon a well known debugger—the GNU *gdb*—to provide conventional debugging commands within each sequential process.

We are also exploiting the fact that most of the basic debugging support functionalities can be integrated into a monitoring layer if it provides well-defined interfaces, and easy definition of new interfaces. This is the only way to offer a stable monitoring layer that will be used by a large number of tools, and will be able to evolve, as new needs arise. Only a few experiments have been proposed towards such goal, namely involving object-based interfaces for event access. A significant and very broad effort has recently been launched towards meeting the above mentioned goals [1]. We are currently pursuing a similar goal, as far as debugging support is concerned, and would like to relate our experimentation with the one being proposed by the OMIS initiative.

3 Conclusions and Ongoing Work

Many existing debugging tools are tied to specific hardware and software environments, or are integrated within specific programming environments. We decided to built the DDBG system due to our goal of investigating issues in the design and implementation of an unified layer for monitoring and debugging.

In summary the current design of the distributed debugging engine provides a resonable flexibility as it allows the user tool, the master daemon, and the remote debuggers to run on distinct machines or processors. On the other hand, our experimentation shows that the current definition of the interface has enough flexibility to accomodate very different tools. For example, our distributed debugging system is currently being used to develop a distributed debugger for PVM-Prolog [11]. PVM-Prolog is an extension to Prolog that provides full access to the PVM environment. Parallel and distributed applications are built by specifying multiple Prolog processes, which cooperate by message-passing primitives. In this implementation, the debugging library is made accessible to each Prolog process through a similar set of basic predicates for process control and inspection.

Ongoing work relates to the implementation of the DDBG architecture, coupled with the support of distributed monitoring functionalities, on top of a heterogeneous distributed architecture consisting of several UNIX nodes, a cluster of ALPHA DEC workstations and two multicomputer machines. The internal architecture of the DDBG system as well as the communication schemes between daemons, debugger instances and application processes will be evaluated and adapted to such environment.

Acknowledgments

This work was partially supported by the CEE COPERNICUS Programme SEPP Project (Contract CIPA-C193-0251) and HPCTI Project (Contract CP-93-5383), by the Portuguese CIENCIA Programme, and PROLOPPE of the PRAXIS XXI Programme, and DEC EERP PADIPRO (Contract no. P-005).

References

[1] T. Ludwig, R. Wismuller, V. Sunderam, A. Bode. *OMIS — on-line monitoring interface specification, Version 1.0.* LRR-TUM, Technical Univ. of Munich, Germany, and Emory Univ. USA, Feb 1996.

[2] University of Westminster, UK. *Software Engineering for Parallel Processing.* Copernicus Programme, Contract CIPA-C193-0251, Progress Report no. 1, Oct 1994.

[3] University of Westminster, UK. *High Performance Computing Tools for Industry.* Copernicus Programme, Contract CP-93-5383, Progress Report no. 3, Apr 1996.

[4] University of Westminster, UK. *Software Engineering for Parallel Processing.* Copernicus Programme, Contract CIPA-C193-0251, Progress Report no. 4, Apr 1996.

[5] G. Dózsa, T. Fadgyas, P. Kacsuk. *GRAPNEL: A Graphical Programming Language for Parallel Programs* Proc. of uP'94: The Eight Symposium on Microcomputer and Microprocessor Applications, Budapest, Hungary, 1994.

[6] H. Krwaczyk, B. Wiszniewski. *Structural Testing of Parallel Software in STEPS* Proc. of the 1st SEIHPC Workshop, COPERNICUS Programme, Braga, Portugal, 1996.

[7] J.C. Cunha, H. Krwaczyk, B. Wiszniewski, P. mork, P. Kacsuk, E. Luque, L. Sutovska, L. Hluchy. *Monitoring and Debugging Distributed Memory Systems.* Proc. of uP'94: The Eight Symposium on Microcomputer and Microprocessor Applications, Budapest, Hungary, 1994.

[8] J.C. Cunha, J. Lourenço, T. Antão. *Integrating a debugging engine to the GRAPNEL environment.* HPCTI Project, COPERNICUS Programme, 3rd Progress Report, University of Westminster, 1996.

[9] J.C. Cunha, J. Lourenço, T. Antão. *A Debugging Engine for a Parallel and Distributed Environment.* Accepted for presentation in the Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS'96), Miskolc, Hungary, October 1996.

[10] A, Beguelin, J.J. Dongarra, G.A. Geist, R. Manchek, V.S. Sunderam. *A User's Guide to PVM Parallel Virtual Machine.* Technical Report, ORNL/TM-118266, Oak Ridge National Laboratory, 1991.

[11] R. Marques, J.C. Cunha. *PVM-Prolog: Parallel Logic Programming in the PVM System.* Procs. of the 1995 PVM User's Group Meeting, Pittsburgh, May 1995.

[12] P. Kacsuk, G. Dózsa, T. Fadgyas. *GRAPNEL: A Graphical Parallel Programming Language.* Journal of Systems Architecture, Special Issue on Parallel Software Engineering, 1996, No. 1.

[13] S. Winter,P. Kacsuk. *Software Engineering for Parallel Processing.* Proc. of the 8th Symp. on Microcomputer and Microprocessor Applications, Budapest, 1994, pp. 285-293.