In Proceedings of 12th International Conference on Software & Systems Engineering and their Applications (ICSSEA 1999), Paris, France, December 1999.

Agent-Aided Software Engineering of High Performance Applications

Junwei Cao
Department of Computer Science, University of Warwick
Gilbert Road, Coventry, CV4 7AL, UK
++44-2476-522485
junwei@dcs.warwick.ac.uk

Abstract

High performance applications are parallel and distributed applications whose performance is usually the most concerned by the developers. Traditional software tools of performance evaluation can not be used efficiently in the Computational Grid, a new software and hardware infrastructure that provides inexpensive access to high-end computational capabilities [3], because software developers of grid-enabled high performance applications may have not much knowledge on parallel and distributed computing and may be not so professional to use current performance modelling and evaluation tools.

In this work the concept of Agent-Aided Software Engineering (AASE) is presented to overcome these difficulties, which integrates autonomous agents with the CASE tools to aid application developers to cope with not only the complexity but also the professionality in software development processes. The Performance Evaluation Agent (PEA) can act autonomously as an expert in the grid application development environment to assist and guide the software development. The main parts of the PEA include an Abstractor, an Evaluator, and an Advisor.

Keywords:

Agent-Aided Software Engineering; High Performance Applications; Performance Evaluation Agent; Computational Grid.

1. Introduction

Computational Grids, composed of distributed and often heterogeneous computing resources, are becoming the platform-of-choice for many performance-challenged applications. A grid is hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [3].

The grid can be described to be four-layered: applications, tools, services, and hardware. The hardware is the infrastructure of the grid, which includes supercomputers interconnected with high-speed networks. The services are the kernel of the grid including performance analysis and scheduling, resource management and allocation, execution monitoring and steering, information management and visualisation, security, accounting, assurance, and so on. Tools are the integrated environments of the APIs of the services and can be used to develop diverse grid-enabled applications. These applications may be distributed, data-intensive, collaborative, or high-throughput, and are all performance-challenged. Thus performance analysis, evaluation and scheduling are essential in order for applications to achieve high performance in the grid environments.

The techniques and tools that are being developed for the performance evaluation of parallel and distributed computing systems are manifold, for example, POEMS [2], AppLeS [11], CHAOS [13], and PACE [7], each having their own motivation and methodology. Though they can all achieve perfect performance evaluation results, they are difficult to be used as tools in the grid environment for the following obvious contradiction:

- The grid, as an infrastructure for high performance computing, aims to enable maximal users to take networking as computing just the same as that have done by the Internet nowadays, as an information infrastructure, which has enabled everyone to use networks to exchange information. So end-users of the grid-enabled applications may have no knowledge on parallel and distributed computing. Developers of applications using the grid-oriented tools may have limited background on performance engineering of software systems.
- Current performance evaluation software tools are not grid-oriented originally. They aim to manage different kinds of the static and dynamic information of the application and the correspondent hardware platform to achieve accurate and effective evaluation results. They are so complex in implementation mechanism that only experts on computing can operate on them.

Agent technology can be exploited to overcome the difficulties. Software agents are software entities that function autonomously in a particular environment [1]. An autonomous agent, called Performance Evaluation Agent (PEA) can act automatically or interactively as an expert on performance evaluation in the development environment of grid-enabled applications to provide the developer with evaluation and analysis results and useful suggestions on how to improve the application performance.

Though agents have been used in many domains like industrial, commercial, medical applications, and entertainment [8], and there have been some agent applications that aim to solve problems in software engineering, the potentiality of using agents to aid software engineering has not been fully aware. In fact, agents can be used in many ways to assist software engineering to cope with diverse problems in all of the life cycles of software development, which we term as Agent-Aided Software Engineering (AASE).

The rest of the paper is organised as follows: Section 2 introduces the concept of AASE. Section 3 gives the structure of the PEA and its implementation mechanism. Preliminary conclusions and future works are discussed in Section 4.

2 Agent-Aided Software Engineering

Computer-Aided Software Engineering (CASE) technologies are tools that aim to reduce the time and cost of software development and enhance the quality of the systems developed. However, the investigation results in [6] show that within the companies that have adopted CASE tools, few people are actually using the tools, and those who use the tools are using few of the functions within the tools. The most important reason is that CASE tools themselves are becoming more and more complex and professional with the increasing of the complexity

of software systems to be developed. New technologies must be integrated into the tools to change the situation. Agent technology can be considered to be one of the choices.

Agent-Aided Software Engineering (AASE) is integrating autonomous agents into CASE tools (or software system developed) to reduce the complexity and professionality of software development.

The research on software engineering before mainly focuses on managing the complexity of the software system and its development. A few agent technologies have been used. For example, in Rational Rose [10], a visual object-oriented modeling tool, the model coherence is maintained by the system automatically. New version of the Internet browser Netscape is integrated with a simple quality feedback agent, which is used to help the developer to get more information on system failures. Also in EDEM platform [4], software agents are used to collect usage data and increase user involvement in the development of interactive systems.

The problem of professionality of software engineering arises especially when the software system with special architecture or requirements is to be developed, for example, parallel and distributed applications. The reason is not only because these kinds of systems are also very complex, but that the developers can be hardly skilled enough to handle correspondent operations. Many ideas from the research of Intelligent User Interface (IUI) can be exploited. Agents can be integrated into the CASE tools interface, which can interact with the application developer, withdraw from the environment or ask the developer for information needed, function as an expert autonomously, and guide the developer with suggestions. In the next section, the Performance Evaluation Agent is introduced to solve the professionality problem in the grid-enabled application development.

3 PEA: the Performance Evaluation Agent

The PEA is a typical example to show the idea of AASE. PEA can be active in the source code editor of the application development environment to provide the developer with performance related information of the application being developed. The structure of PEA is illustrated in Figure 1, which mainly includes three parts: an Abstractor, an Evaluator, and an Advisor.

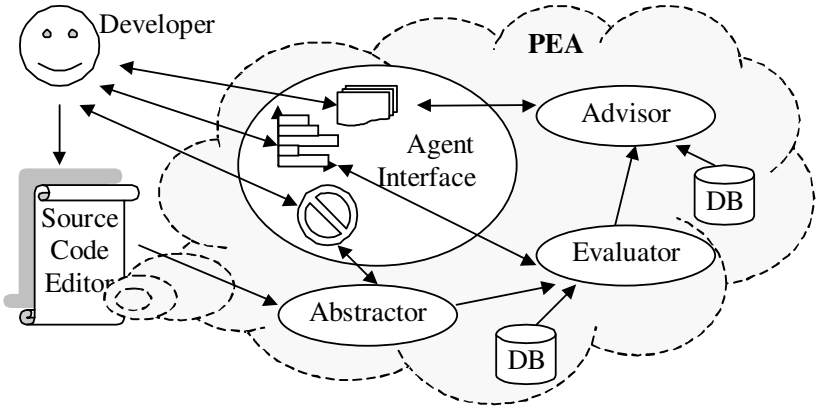

Figure 1. PEA Structure: Abstractor, Evaluator, and Advisor

As shown in the figure, the agent interfaces are used for the PEA to interact with the developer in different ways. The arrows indicate the directions of data flows. Several databases are used to provide the agent with correspondent information. Since the application development tools are parts of the grid environment, the implementation of the Abstractor, Evaluator, and Advisor of the PEA may rely on other services provided by the grid. The structures and functions of them are introduced respectively in greater details below.

3.1 Abstractor

The Abstractor is responsible for collecting resource usage information from the developer, the grid environment and the application source codes, and transforming them into performance models that can be used by the Evaluator. The methodologies of current performance evaluation tools can be exploited to implement the Abstractor. As shown in Figure 2, there are three main activities in this process of abstraction: separation, transformation, and compilation.

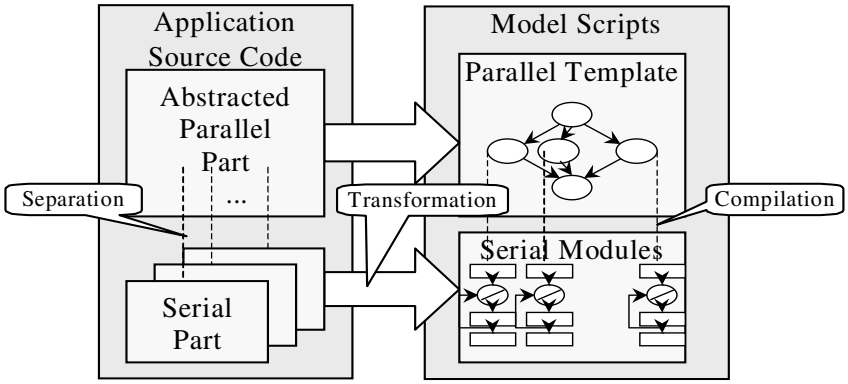

Figure 2. PEA Performance Model Abstraction

- *Separation*. The methodology to implement the Abstractor is basically the separation of the communication and computation of parallel and distributed program. Unfortunately, most of the current programming tools, such as PVM and MPI, do not support syntactic separation of sequential parts and parallel parts of the program. So the computation of the program should be abstracted into sequential functions respectively and the abstracted program can be taken as parallel templates that describe the parallelism of the application.

- *Transformation*. The kernel of the Abstractor is the transformation of the separated programs into correspondent resource usage specification of the application, which can be called Performance Specification Language (PSL). For example, CHIP^{3}S [9] is such a PSL that has been used as the kernel of the PACE system. The resource usage concerned is usually the time-consuming (CPU time and elapse time). Each part of the program must be transform to be one module of the performance model in PSL. The resource usage specifications for computation and communication are different. The CPU time for computation is related to the type of the programming language (e.g. C and Fortran) and the instructions used; The time-consuming for communication is related to the type of the parallel programming interface (e.g. PVM and MPI) used and the message size. The process of the transformation may be semi-automatic because the dynamic information of the program, such as the loop number and the execution probability of the *if* statement, must be provided by the developer.

- *Compilation*. The modules transformed from the separated program are compiled into one performance model to give a global view of the resource usage of the application, which can be used by the Evaluator.

The performance modeling process can be performed automatically, which is paramount important for the PEA to achieve the autonomy. The agent only asks the developer very simple questions through the interactive interface and hides a lot of technical details.

3.2 Evaluator

The Evaluator is responsible for the evaluation of the performance model according to the developer's requirement and provides the developer with application performance information. Most of the evaluation engines in current tools can not work on the fly, so can not be used in the PEA. The evaluation process in the PEA must be quick enough for the agent to achieve autonomy. This can be achieved by providing the Evaluator with a database of hardware models, which can be maintained off-line.

The application performance is related to not only the software models but also the hardware platform. For example, in PACE system, Hardware Model Configuration Language (HMCL) allows specifying system-dependent parameters. On evaluation, the relevant sets of parameters are used, and supplied to the Evaluator for each of the component models. An example hardware model is shown in Figure 3 illustrating the main subdivision currently considered involving a distinction between computation, communication, memory and I/O models.

The data of the hardware models can be collected off-line using the special benchmark programs. The abundant off-line configuration information included by the hardware model database is the basis to implement a rapid evaluation time to produce the performance evaluations. The hardware models can be changed

according to the requirement of the developer (even cross-platform performance evaluation can also be achieved). The evaluation results can be shown directly to the developer via the agent interface, or they can be analysed and compared, which is done by the Advisor.

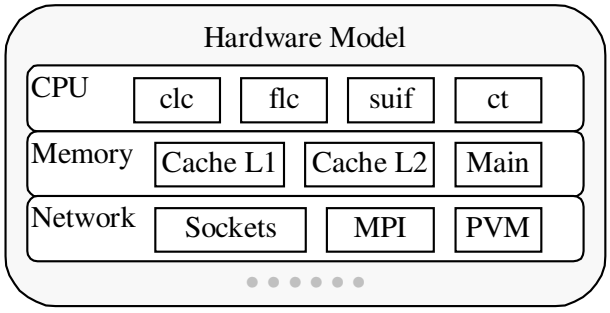

Figure 3. Hardware Model

3.3 Advisor

The Advisor is responsible to analyze raw evaluation results and guide the software development in a more concrete way. Simple analysis can be processed.

Evaluation results on time and cost on different hardware platform with different number of processors can be compared to show which is more optimum. The developer may think naively that run time will decrease when the number of processors increases. The actual situation can rather different. Sometimes the run time may be bound to a constant or even increase with the increasing of the number of processors. In the grid environment, many computing capabilities can be choosed, some of them may be more powerful and, thus, more expensive to use. So there is also the issue of optimization.

Bottleneck analysis can be done if provided more details of evaluation results on the workload of each processors. The workload on different processors may be unbalance and some processors may wait too much time for others to finish. The Advisor may tell the developer to improve the software architecture or parallel strategy, or suggest the developer to replace the bottleneck processor with more powerful one.

AI techniques like Expert System (ES) can be added to the implementation of the Advisor to do further analysis, such as fault-tolerance and deadlock. This is out scope of the current considered research.

4 Conclusions and Future Works

The research on Agent-Aided Software Engineering is just beginning. Researchers of software engineering have not been fully aware of the potentialities of using agent technology to assist software development.

In this work, agent technologies are attempted to be integrated into CASE tools and help to solve the problems of complexity and professionality of the Grid-enabled application development. The Performance Evaluation Agent mainly includes an Abstractor, an Evaluator, and an Advisor, which can act as an expert autonomously in the grid application development environment. Many further works can be done on basis of current implementation of the PEA.

The agent interface should be more flexible to interact with the developer. Many dynamic information in the source code of the application can not be withdrawn automatically and can only be provided by the developer directly. Multimedia technologies can be used to make the interface much more effective and friendly as what has been done in some pedagogical agents. For example, in Adele architecture [12], an animated persona can support continuous multi-modal interaction with a student.

The PEA Advisor is not easy to be practical. Many current analyses are done by experienced expert. There are also some analogous advisor systems in agent applications on other fields, which should be referred to. For example, Guardian [5] is an autonomous agent for medical monitoring and diagnosis, in which several algorithms cooperate to produce diagnoses and treatment plans under hard real-time conditions.

The Computational Grid will be a new infrastructure in the next millennium, which will enable many new kind of application. However, the software engineering tools can not even be used to fulfil the current

development of software system. New technologies should be developed to meet the ever-increasing needs.

References

[1] J. M. Bradshaw (ed). Software Agents. The AAAI Press/The MIT Press, 1997.

[2] E. Deelman, A. Dube, A. Hoisie, Y. Luo, R. L. Oliver, D. Sundaram-Stukel, H. Wasserman, V. S. Adve, R. Bagrodia, J. C. Browne, E. Houstis, O. Lubeck, J. Rice, P. J. Teller, and M. K. Vernon. POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational Systems. Proceedings of the ACM 1st International Workshop on Software and Performance, pp. 18-30, 1998.

[3] I. Foster, and C. Kesselman. The GRID: Blueprint for a New Computing Infrastructure. Morgan-Kaufmann, July 1998.

[4] D. M. Hilbert, J. E. Robbins, and D. F. Redmiles. EDEM Intelligent Agents for Collecting Usage Data and Increasing User Involvement in Development. Proceedings of the 1998 International Conference on Intelligent User Interfaces, 1998, pp. 73-76.

[5] J. E. Larsson, and B. Hayes-Roth. Guardian: An Intelligent Autonomous Agent for Medical Monitoring and Diagnosis. IEEE Intelligent Systems & Their Applications, 13(1), January/February 1998.

[6] D. Lending, and N. L. Chervany. The Use of CASE Tools. Proceedings of the ACM SIGCPR Conference on Computer Personnel Research, 1998, pp. 49-58.

[7] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems. To appear in High Performance Systems, Sage Science Press, 1999.

[8] N. R. Jennings and M. J. Wooldridge (eds). Agent Technology: Foundations, Applications, and Markets. Springer-Verlag, 1998.

[9] E. Papaefstathiou, D. J. Kerbyson, G. R. Nudd, and T. J. Atherton. An Overview of the CHIP^{3}S Performance Prediction Toolset for Parallel Systems. Proceedings 8th ISCA International Conference on Parallel and Distributed Computing Systems, Orlando, 1995, pp. 527-533.

[10] T. Quatrani. Visual Modeling with Rational Rose and UML. Addison-Wesley Object Technology Series, 1998.

[11] J. M. Schopf. Structural Prediction Models for High-Performance Distributed Applications. Proceedings of 1997 Cluster Computing Conference (CCC'97), Atlanta, 1997.

[12] E. Shaw, W. L. Johnson, and R. Ganeshan. Pedagogical Agents on the Web. Proceedings of 3rd International Conference on Autonomous Agents, pp. 283-290.

[13] M. Uysal, T. M. Kurc, A. Sussman, and J. Saltz. A Performance Prediction Framework for Data Intensive Applications on Large Scale Parallel Machines. Proceedings of the 4th Workshop on Languages, Compilers and Run-time Systems for Scalable Computers (LCR'98), 1998.