Teuta: Tool Support for Performance Modeling of Distributed and Parallel Applications*

Thomas Fahringer[1], Sabri Pllana[2], and Johannes Testori[2]

[1] Institute for Computer Science, University of Innsbruck
Technikerstraße 25/7, 6020 Innsbruck, Austria
Thomas.Fahringer@uibk.ac.at
[2] Institute for Software Science, University of Vienna
Liechtensteinstraße 22, 1090 Vienna, Austria
{pllana,testori}@par.univie.ac.at

Abstract. In this paper we describe Teuta, which we have developed to provide tool support for the UML-based performance modeling of distributed and parallel applications. Teuta features include model checking and model traversing. Model checking is used to verify whether the model conforms to the UML specification. In addition, Teuta supports semantic model checking for the domain of high performance computing. For the generation of different model representations the model traversing is used. In addition, we present our methodology for automatic generation of the simulation model from the UML model of an application. This simulation model is used to evaluate the performance of the application. We demonstrate the usefulness of Teuta by modeling LAPW0, a distributed material science application.

Keywords: Distributed and Parallel Applications, Modeling and Simulation, Performance, UML

1 Introduction

The high performance computing is usually used for solving complex problems in science and engineering. However, effective performance-oriented program development requires the programmer to understand the intricate details of the programming model, the parallel and distributed architecture, and the mapping of applications onto architectures. On the other hand, most performance tools [3, 4] provide little support at early application development stages when several designs and strategies are examined by an application programmer. We anticipate that a tool which supports the application programmer to graphically develop the performance model of application at an early development stage would help to influence design decisions without time-consuming modifications of the code of an already implemented application.

* The work described in this paper is supported in part by the Austrian Science Fund as part of Aurora Project under contract SFBF1104.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 456–463, 2004.

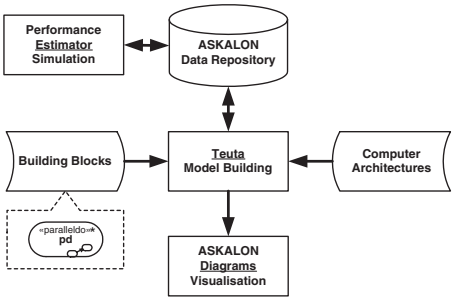


Fig. 1. Performance Prophet architecture

In this paper we give an overview of our tool *Teuta*, which supports the development of performance models for parallel and distributed applications based on the Unified Modeling Language (UML) [5]. The UML is de facto standard visual modeling language which is a general purpose, broadly applicable, tool supported, industry standardized modeling language. However, by using only the core UML, some shared memory and message passing concepts can not be modeled in an adequate manner. In order to overcome this issue we have developed an extension of UML for the domain of high performance computing [6].

Teuta is an integral part of our performance prediction system *Performance Prophet* [1]. Figure 1 depicts the architecture of Performance Prophet. The role of Teuta is to assist the user to develop the model for an application by composing existing *building blocks*. The application model is enriched with cost functions. Thereafter, Teuta transforms the annotated model to an intermediate form based on which the *Performance Estimator* of Performance Prophet evaluates the performance of the application on the computer *architecture* selected by user.

Teuta features include model checking and model traversing. Model checking is used to verify whether the model conforms to the UML specification. In addition, Teuta supports semantic model checking for the domain of High Performance Computing. Teuta makes use of model traversing for the generation of different model representations (such as XML, C++).

Furthermore, in this paper we present our methodology for automatic generation of the simulation model from the UML model of an application. This simulation model is used to evaluate the performance of the application. We demonstrate the usefulness of Teuta and our methodology by modeling LAPW0, a distributed material science application.

The rest of this paper is organized as follows. An overview of Teuta is described in Section 2. Section 3 presents our methodology for the automatic generation of the simulation model based on the UML model of an application. A case study is described in Section 4. Finally, some concluding remarks are made and future work is outlined in Section 5.

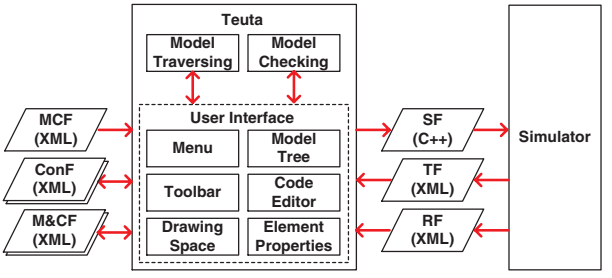


Fig. 2. The architecture of Teuta. Description of abbreviations: MCF - Model Checking File, ConF - Configuration Files, M&CF - Model and Construct Files, SF - Simulation File, TF - Trace File, RF - Result File.

2 An Overview of Teuta

Teuta[1] is a platform independent tool for UML-based modeling of parallel and distributed applications.

Because the application developers may work on various platforms, the tool should be able to run on various platforms as well. Therefore, we have used the Java language for the implementation of Teuta, based on the Model-View-Controller (MVC) paradigm [2]. MVC is a paradigm that enforces the separation between the user interface and the rest of the application. In this paradigm, the *model* represents data and the core functionality of the application. The *view* is the user interface. The *controller* interprets and delegates the interactions with the *view* (for instance button clicks) to the *model*, which performs the corresponding actions. The advantage of MVC is that it allows creation of independent *views* (user interfaces) that correspond to a single *model* of an application.

It is difficult to foresee all the types of the building blocks that the user might want to use to model his application. Therefore, we have included a basic set of building blocks, which are described in [6,7], and made it easy to extend the tool with the new building blocks. Teuta may be extended with new building blocks by modifying a set of XML-based configuration files.

Figure 2 shows the architecture of Teuta, which consists of three main parts: (i) Model checking; (ii) Model traversing; (iii) Graphical user interface (GUI). We will describe each of these parts in the subsequent sections.

Element *M&CF* in Figure 2 indicates XML files which contain application model and constructs. The application model is stored in a file, which contains all the information which is required to display the model in the editor. All the diagrams and modeling elements with their properties and geometric information are stored in this file. A construct is a set of modeling elements. For instance a set of modeling elements that represents a loop. Constructs may be used to simplify the development of the application model.

Element *ConF* in Figure 2 indicates XML files which are used for configuration of Teuta.

[1] Teuta (Tefta): Queen of the Illyrians (231-228 BC)

The communication between Teuta and the simulator is done via files *SF*, *TF*, and *RF* (see Fig. 2). Element *SF* indicates the C++ representation of the application performance model, which is generated by Teuta. Elements *TF* and *RF* represents the trace file and result file respectively, which are generated by the simulator.

Element *MCF* in Figure 2 indicates the XML file, which is used for model checking. The following Section describes the model checking in more detail.

2.1 Model Checking

This part of Teuta is responsible for the correctness of the model. The rules for model checking are specified by using our XML based Model Checking Language (MCL). The model checker gets the model description from an MCL file. This MCL file contains a list of available diagrams, modeling elements and the set of rules that defines how the elements may be interconnected.

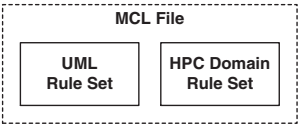


Fig. 3. Model checking rule sets

Based on the *UML Rule Set* the model checker verifies whether the application model conforms to the UML specification (see Fig 3). In addition, Teuta supports *semantic model checking* for the domain of High Performance Computing (HPC). The *HPC Domain Rule Set* specifies whether two modeling elements can be interconnected with each other, or nested one within another, based on their semantics. For instance, it is not allowed to place the modeling element *BARRIER* within the modeling element *CRITICAL*, because this would lead to the deadlock.

2.2 Model Traversing

This component of Teuta provides the possibility to walk through the model, visit each modeling element, and access its properties (for instance element name). We use the model traversing for the generation of various model representations (such as XML and C++).

The *Model Traversing* component of Teuta consists of three parts (i) the *Navigator*, (ii) the *Traverser*, (iii) and the *ContentHandler*.

The *Navigator* is responsible for interpreting the input, creating the corresponding object model and for the navigation in this object model.

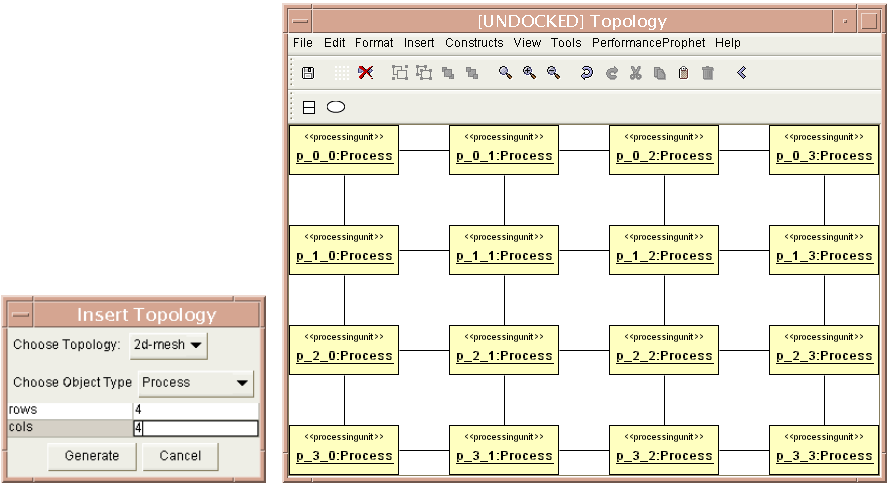
The *Traverser* is responsible for traversing the object model created by the *Navigator*. It walks through the diagrams of the model by calling the *Navigator's* methods. Various types of traversing strategies can be implemented, such as depth-first or breadth-first.

The *ContentHandler* methods are called by the *Traverser* whenever the *Traverser* begins or finishes a model, or visits an element. It is responsible for the output of the traversing, for instance it can create another object model or write to a file in a specific format.

2.3 Graphical User Interface (GUI)

Figure 6(a) in Section 4 depicts Teuta GUI, which consists of: *menu*, *toolbar*, *drawing space*, *model tree*, *code editor*, and *element properties*. Because of the limited space we are not able to describe the graphical user interface of Teuta in detail. Therefore, in this section we describe only the support of Teuta for constructs. A construct is a set of modeling elements. The idea is to relinquish the user from trivial time consuming tasks, by automatic generation of constructs. We have identified two types of constructs: (i) simple constructs that are used frequently, such as loops; (ii) large constructs with regular structure, such as topologies of processes.

Figure 4 shows an example of the automatic generation of topology of processes. The user specifies the type (for instance 2D mesh) and parameters (for instance 4 rows, 4 columns) of the process topology (see Figure 4(a)). Based on this information Teuta generates the corresponding process topology (see Figure 4(b)). The process topology is represented by the UML CollaborationInstanceSet (see [7]). Without this support of Teuta, the user would spend a significant amount of time to create all the modeling elements and arrange them to represent the process topology.



(a) Specification (b) Topology of processes

Fig. 4. An example of the automatic generation of topology of processes

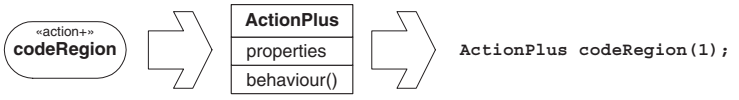


Fig. 5. From the UML model to the simulation model

3 The Development of the Simulation Model Based on the UML Model

Figure 5 depicts the process of transition from the model of the application represented by UML activity diagram, to the simulation model. The simulation model is represented in C++.

The UML element *activity* is stereotyped as *action+*. The UML stereotyping extension mechanism makes possible to make the semantics of a core modeling element more specific (see [6]). The element *action+* is used to represent a code region of an application. For this modeling element we have defined the class *ActionPlus*. The properties of the UML modeling element *action+* (for instance the element ID) are mapped to the *properties* of the class *ActionPlus*. The performance behavior of the element is defined in the method *behaviour()* of the class. This method is responsible for advancing the time in the simulation clock. This time is estimated either by a parameterized cost function or by simulating the behavior of the element. The name of the UML modeling element, in our example *codeRegion*, is mapped to the name of the instance of the class.

In the next section we present the performance modeling and evaluation of a real-world application.

4 Case Study: LAPW0

The objective of our case study is to examine whether the tool support described in this paper is sufficient to build a performance model for a real-world application. The application for our study LAPW0, which is a part of the WIEN2k package [9], was developed at Vienna University of Technology. The Linearized Augmented Plane Wave (LAPW) method is among the most accurate methods for performing electronic structure calculations for crystals. The code of LAPW0 Application is written by using FORTRAN90 and MPI.

The LAPW0 application consists of 100 file modules (a module is a file containing source code). The modeling procedure aims to identify the more relevant (from performance point of view) code regions. We call these code regions building blocks. A building block can be a sequence of computation steps, communication operations or input/output operations. In order to evaluate the execution time of the relevant code regions of LAPW0 application, we have instrumented these code regions and measured their corresponding execution times by using SCALEA [11], which is a performance analysis tool for distributed and parallel applications.

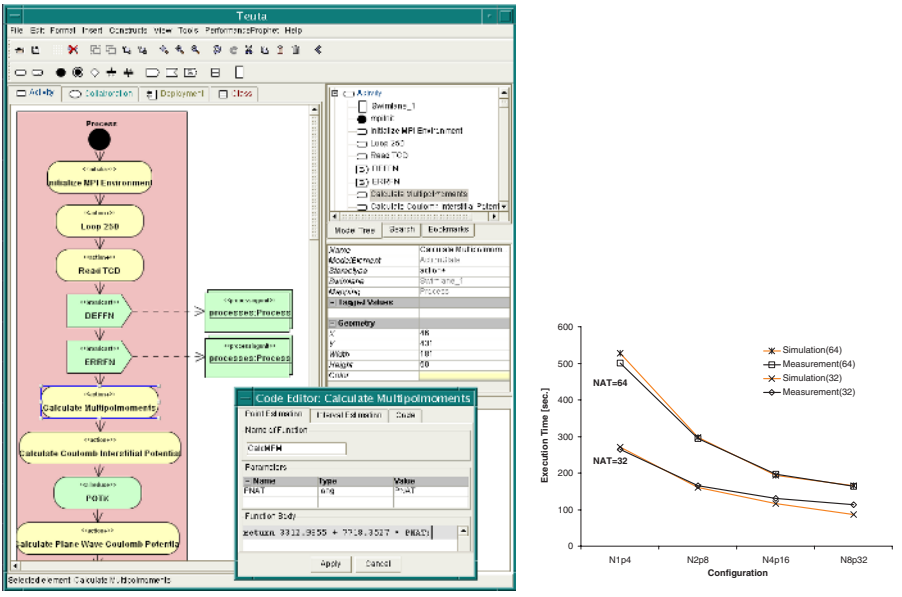


Fig. 6. Performance modeling and evaluation of the LAPW0 application

Figure 6(a) depicts the model of LAPW0, which is developed with Teuta. Due to the size of the LAPW0 model, we can see only a fragment of the UML activity diagram within the drawing space of Teuta. On the right hand side of Figure 6(a) is shown how the model of LAPW0 is enriched with cost functions by using Teuta *code editor*. A cost function models the execution time of a code region.

In order to evaluate the model of LAPW0 we have transformed the high level UML model of LAPW0 into a simulation model. Teuta automatically generates the corresponding C++ representation, which is used as input for the *Performance Estimator* (see Figure 1). The Performance Estimator includes a simulator, which models the behavior of cluster architectures. CSIM [10] serves as a simulation engine for the Performance Estimator.

Figure 6(b) shows the simulation and measurement results for two problem sizes and four machine sizes. The problem size is determined by the parameter NAT, which represents the number of atoms in a unit of the material. The machine size is determined by the number of nodes of the cluster architecture. Each node of the cluster has four CPU's. One process of the LAPW0 application is mapped to one CPU of the cluster architecture. The simulation model is validated by comparing the simulation results with the measurement results. We consider that this simulation model provides the performance prediction results with the accuracy which would be sufficient to compare various designs of the application.

5 Conclusions and Future Work

In this paper we have described Teuta, which provides the tool support for the UML-based performance modeling of parallel and distributed applications. We have demonstrated the usefulness of Teuta by modeling LAPW0, which is a distributed material science application. Based on the high level UML model of LAPW0 application the simulation model is automatically generated by Teuta. The simulation model is validated by comparing the simulation results with the measurement results. We consider that this simulation model provides the performance prediction results with the accuracy which would be sufficient to compare various designs of the application.

Currently, we are extending Teuta for modeling Grid [8] applications.

References

1. T. Fahringer and S. Pllana. Performance Prophet. University of Vienna, Institute for Software Science. Available online: http://www.par.univie.ac.at/project/prophet.
2. G. Krasner and S. Pope. A cookbook for using the Model-View-Controller interface paradigm. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.
3. D. Kvasnicka, H. Hlavacs, and C. Ueberhuber. Simulating Parallel Program Performance with CLUE. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 140–149, Orlando, Florida, USA, July 2001. The Society for Modeling and Simulation International.
4. N. Mazzocca, M. Rak, and U. Villano. The Transition from a PVM Program Simulator to a Heterogeneous System Simulator: The HeSSE Project. In *7th European PVM/MPI*, volume 1908 of *Lecture Notes in Computer Science*, Balatonfüred, Hungary, September 2000. Springer-Verlag.
5. OMG. Unified Modeling Language Specification. http://www.omg.org, March 2003.
6. S. Pllana and T. Fahringer. On Customizing the UML for Modeling Performance-Oriented Applications. In *<<UML>> 2002, "Model Engineering, Concepts and Tools", LNCS 2460, Dresden, Germany*. Springer-Verlag, October 2002.
7. S. Pllana and T. Fahringer. UML Based Modeling of Performance Oriented Parallel and Distributed Applications. In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, California, USA, December 2002. IEEE.
8. S. Pllana, T. Fahringer, J. Testori, S. Benkner, and I. Brandic. Towards an UML Based Graphical Representation of Grid Workflow Applications. In *The 2nd European Across Grids Conference*, Nicosia, Cyprus, January 2004. Springer-Verlag.
9. K. Schwarz, P. Blaha, and G. Madsen. Electronic structure calculations of solids using the WIEN2k package for material sciences. *Computer Physics Communications*, 147:71–76, 2002.
10. Herb Schwetman. Model-based systems analysis using CSIM18. In *Winter Simulation Conference*, pages 309–314. IEEE Computer Society Press, 1998.
11. Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Distributed and Parallel Program. In *8th International Europar Conference(EuroPar 2002)*, Lecture Notes in Computer Science, Paderborn, Germany, August 2002. Springer-Verlag.