A Framework for an Interoperable Tool Environment

Radu Prodan[1] and John M. Kewley[2]

[1] Institut für Informatik, Universität Basel,
Mittlere Strasse 142, CH-4056 Basel, Switzerland
`prodan@ifi.unibas.ch`
[2] Centro Svizzero di Calcolo Scientifico (CSCS),
Via Cantonale, CH-6928 Manno, Switzerland.
`kewley@cscs.ch`

Abstract. Software engineering tools are indispensable for parallel and distributed program development, yet the desire to combine them to provide enhanced functionality has still to be realised. Existing tool environments integrate a number of tools, but do not achieve interoperability and lack extensibility. Integration of new tools can necessitate the redesign of the whole system. We describe the FIRST tool framework, its initial tool-set, classify different types of tool interaction and describe a variety of tool scenarios which are being used to investigate tool interoperability.

1 Introduction

A variety of software engineering tools are now available for removing bugs and identifying performance problems, however these tools rely on different compilation options and have no way of interoperating. Integrated tool environments containing several tools do offer some degree of interoperability; they do, however, have the disadvantage that the set of tools provided is fixed and the tools are typically inter-dependent, interacting through internal proprietary interfaces. The result is a more complex tool which combines the functionality of the integrated tools, but lacks true interoperability and is closed to further extension.

The FIRST[1] project, a collaboration between the University of Basel and the Swiss Center for Scientific Computing, defines an extensible framework [1] for the development of interoperable software tools. Tool interaction is enhanced by its high-level interoperable tool services [2], which provide a tool development API. Figure 1 shows the object-oriented architecture for FIRST (see [2] for further information). The Process Monitoring Layer (PML) handles the platform dependent aspects of the tool; it attaches to an application and extracts run-time information using the Dyninst [3] dynamic instrumentation library. The Tool Services Layer utilises this functionality to present a set of high-level tool services

[1] Framework for Interoperable Resources, Services, and Tools.
Supported by grant 21–49550.96 from the Swiss National Science Foundation.

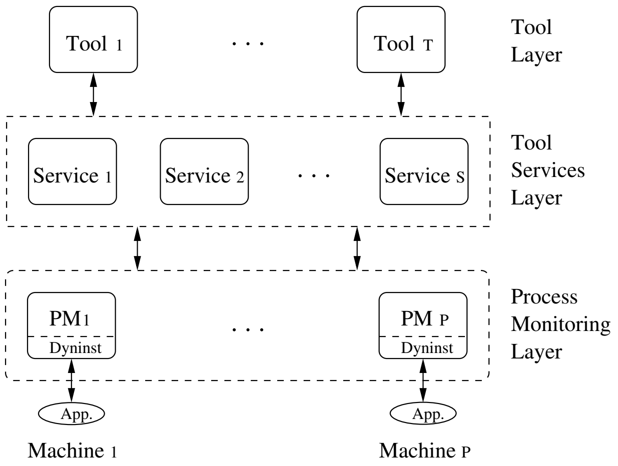

Fig. 1. The FIRST Architecture.

to the tool developer: *Information Requests* provide information about the state of the running application. *Performance Metrics* specify run-time data (e.g., counters and timers), to be collected by the PML; *Breakpoints* set normal or conditional breakpoints in the application; and *Notifications* inform the requester when certain events have occurred.

This paper classifies different types of tool interaction and describes a variety of tool scenarios (Sec. 3) to be used to investigate tool interoperability.

2 Initial Toolset

The FIRST project focuses on the design and development of an open set of tool services, however to prove the viability of the framework, a set of interoperable tools has been implemented. These tools operate on unmodified binaries at run-time and can be used to monitor both user and system code even when there is no source code available. There is no dependence on compilation options or flags. The intention is to develop simple tools that, as well as demonstrate the concepts of the FIRST approach, can then be used as building blocks that interoperate to reproduce the functionality of traditionally more complex tools.

`Object Code Browser (OCB)` is a browsing tool, which graphically displays the object code structure of a given process. It can also be used alongside other tools for selecting the functions to be instrumented (Sec. 3.1).

`1st_top` inspired by the Unix administration tool `top`, was designed to display the top n functions in terms of the number of times they were called, or in terms of execution time.

`1st_trace` in the style of the Unix software tool `truss`, traces the functions executed by the application as it executes.

`1st_prof` like the Unix tool `prof` displays the call-graph profile data, by timing and counting function calls.

`1st_cov` imitates the Unix tool `tcov` to produce a test coverage analysis on a function basis.

`1st_debug` is a traditional debugger, like `dbx`, with the capabilities of controlling processes, acquiring the object-structure of the application, viewing variables, and adding breakpoints.

This set of tools will be extended to cover more specific parallel programming aspects (e.g., deadlock detector, message queue visualiser, distributed data visualiser, and memory access tool).

3 Tool Interoperability Scenarios

One of the main issues addressed by the framework is to provide an open environment for interoperability and to explore how various tools can co-operate in order to gain synergy. FIRST considers several types of tool interaction [4]:

Direct Interaction assumes direct communication between tools. They are defined by the tool's design and implementation, and happen exclusively within the tool layer; they are not performed by the framework.

Indirect Interaction is a more advanced interaction, mediated by the framework via its services. It requires no work or knowledge from the tools (which might not even know about each others existence). It occurs when the FIRST tool services interact with each other on behalf of the tools. These indirect interactions can occur in several ways:

Co-existence when multiple tools operate simultaneously, each on its own parallel application, sharing FIRST services,

Process Sharing when multiple tools attach and instrument the same application process at the same time. This kind of interoperability is consistently handled by the framework through its co-ordinated services.

Instrumentation Sharing when tools share instrumentation snippets while monitoring the same application process to minimise the probe-effect. This is automatically handled by FIRST.

Resource Locking when tools require exclusive access to a specific resource. For example a tool could ask for a lock on a certain application resource (process or function) so that it may perform some accurate timing. No other tool would be allowed to instrument that resource, but could instead use the existing timers (instrumentation sharing).

3.1 Interaction with a Browser

Many tools need to display the resource hierarchy [5] of an application. This includes object code structure (modules, functions and instrumentation points), machines, processes and messages. Since it is inefficient for every tool to independently provide such functionality, the responsibility can be given to a single tool, the OCB (Sec. 2). This can display the resource hierarchy of an application and also be used to specify which resources are to be used when another FIRST tool is run. For instance by selecting a set of functions and then running `1st_prof` with no other arguments, the selected functions will be monitored.

3.2 Computational Steering

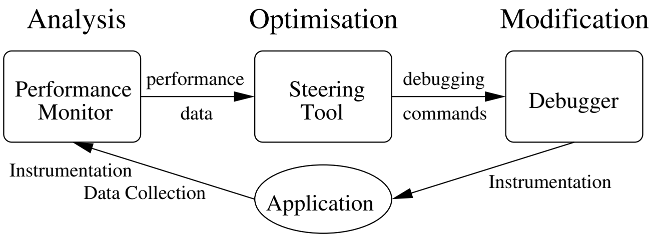

Fig. 2. The Steering Configuration.

The FIRST computational steering tool will directly interact with two other tools: a performance monitor which collects performance data and presents it to the steering tool and a debugger which dynamically carries out the optimisations chosen by the steering tool. The use of dynamic instrumentation enables the steering process to take place dynamically, within one application run, hence there is no need to rerun the application every time a modification is made.

The interoperability between the three tools is mixed. The steering tool interacts directly with the other two. The performance monitor and the debugger interact indirectly, by concurrently manipulating the same application process (using the same monitoring service).

3.3 Interaction with a Debugger

The interaction of tools with a run-time interactive debugger requires special attention due to its special nature of manipulating and changing the process's execution. Two distinct indirect interactions (process sharing) are envisaged:

Consistent Display Providing a consistent display is an important task no run-time tool can avoid. When multiple tools are concurrently monitoring the same processes, this issue becomes problematic since each tool's display depends not only on its own activity, but also on that of the others'.

When a visualiser interoperates with a debugger the following interactions could happen:

– if the debugger stops the program's execution, the visualiser needs to update its display to show this;
– if the debugger changed the value of a variable, for consistency, the visualiser should update its display with the new value;
– if the debugger loaded a shared library in the application, or replaced a call to a function, the OCB must change its code hierarchy accordingly.

Timing Another important interaction can happen between a performance tool and a debugger. For example, a debugger could choose to stop a process whilst the performance tool is performing some timing operations on it. Whilst the user and system time stop together with the process, the wall-time keeps running. The framework takes care of this and stops the wall-timers whenever the application is artificially suspended by the debugger.

4 Conclusion and Future Work

This paper has introduced an open extensible framework for developing an interoperable tool environment. The requirements for a set of interoperable tools were described along with a brief introduction to the software tools currently available within FIRST. The notion of tool interoperability was analysed and the resulting classification of tool interactions was presented. Interoperability scenarios were presented to investigate the various ways tools can interact to gain synergy.

These scenarios for interoperating tools will be the subject of future work to validate the capacity of the framework for interoperability.

References

[1] Radu Prodan and John M. Kewley. FIRST: A Framework for Interoperable Resources, Services, and Tools. In H. R. Arabnia, editor, *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (Las Vegas, Nevada, USA)*, volume 4. CSREA Press, June 1999.
[2] John M. Kewley and Radu Prodan. A Distributed Object-Oriented Framework for Tool Development. In *34th International Conf. on Technology of Object-Oriented Languages and Systems (TOOLS USA)*. IEEE Computer Society Press, 2000.
[3] Jeffrey K. Hollingsworth and Bryan Buck. DyninstAPI Programmer's Guide. Manual, University of Maryland, College Park, MD 20742, September 1998.
[4] Roland Wismüller. Interoperability Support in the Distributed Monitoring System OCM. In R. Wyrzykowski et al., editor, *Proceedings of the 3rd International Conf. on Parallel Processing and Applied Mathematics (PPAM'99)*, volume 4, pages 77–91. Technical University of Czestochowa, Poland, September 1999.
[5] Barton P. Miller, R. Bruce Irvin, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, Karen L. Karavanic, Krishna Kunchithapadam, and Tia Newhall. The Paradyn Parallel Performance Measurement Tool. *IEEE Computer*, 28(11):37–46, November 1995.