

Developing an Open Architecture for Performance Data Mining

David B. Pierce¹ and Diane T. Rover²

¹ MS 1C1, Smiths Industries, 3290 Patterson Ave SE, Grand Rapids, MI, 49512
pierce_david@si.com

² Dept. of Elec. and Computer Engineering, Michigan State Univ., E. Lansing, MI, 48824
rover@egr.msu.edu

Abstract. Performance analysis of high performance systems is a difficult task. Current tools have proven successful in analysis tasks but their implementation is limited in several respects. Closed architectures, predefined analysis and views, and specific platforms account for these limitations. Embedded systems are particularly affected by these concerns. This paper presents an open architecture for performance data mining that addresses these limitations. Comparisons of the architecture with current tools show its capabilities address a wider range of system phases and environments.

1 Introduction

Performance analysis of complex systems is a difficult task. As a result, methods and tools to manage and reduce performance data to useable quantities or useful representations are the focus of significant research. Some successful tools include Pablo [1], Paragraph [2], and SPI [3]. These tools receive events from files, embedded instrumentation, or from an Instrumentation System (IS). These tools generally have a predefined set of views selected from a menu, some have options to select data to display, and libraries or executables to compile the tool.

Despite their successes in the lab environment, this class of tools are not an integral part of embedded and high performance systems because:

- The usage environment is limited to a specific OS or target HW,
- The design/source is protected or incomplete, limiting ability for integration,
- The views, processing algorithms, and queries (some tools have no query mechanism) are predefined, limiting flexibility for specific problems,
- The data sources/sinks are limited, limiting the use of the system and its results.

These tools are geared toward a lab environment, but we want to extend performance analysis to other environments. This will support embedded high performance systems, which can utilize performance analysis results for greater efficiency, user directed fault tolerance, and environmental tolerance (the recognition and corrective action operational conditions exceeding worst case design scenarios).

A solution to these limitations is to define a Performance Data Mining Architecture (PDMA) that: 1) has an open architecture described in a format consistent with a wide range of system design tools, 2) that addresses the data mining capabilities needed for large quantities of data, and 3) is flexible and extensible (concerning views, algorithms, queries, data exchange, and data storage) allowing for a wide range of systems and interfaces, and further development of the individual pieces as specific systems and applications dictate.

This paper presents the definition of such an architecture, with comparisons to current tools showing the benefits and advantages of such an approach.

2 Unified Modeling Language

To enable the widespread use of a PDMA in system designs, the development of a PDMA must address the system design environment. System designs are documented, reviewed, and analyzed in the early stages through the use of modeling techniques, such as Structured Analysis and UML. Following stages of system design use the model created to generate requirements, test plans and procedures, and in some cases, to generate source code headers and/or source code. While the most effective technique is a subject for debate [4], the great utility of these methods is not.

We have utilized the Unified Modeling Language [5] (UML) for the development of the PDMA. UML is widely accepted within the systems community, and its usage is increasing [6]. By using UML, the design of a PDMA is expressed in the same format as the system design itself, promoting ready implementation into design, analysis, test, and documentation. There are a significant number of tools that can analyze, simulate, and generate code from suitable UML diagrams, securing a spot for data mining at the ground floor of a system, and making it one of the important features of a complete system.

3 A Performance Data Mining Architecture

To begin, we examine current performance analysis tools, which have been successful in at least one phase of a system lifespan. These tools have one or more common tasks: 1) data input (performance events or statistics), 2) computation of statistics or data points, 3) display of data, and 4) user interface. A query function is also present on a few tools.

These common tasks summarize a significant portion of the desired system. However, there are four additional tasks that extend the usage environment of these tools. First, a database function to provide more flexibility to queries and more support for long term or relational computations is needed. Second, an output function for an IS, providing the ability to change instrumentation, based on current data. Third, provision for data exchange with system applications is important, and will support

contextual analysis of the system model, requirements, and testing in a wide range of operational environments.

These common tasks then comprise the Use Cases within the Use Case diagram and form the basic requirements. The Use Case diagram is shown in Fig. 1. There is much detail underlying these simple use cases that differentiate the desired architecture from existing tools.

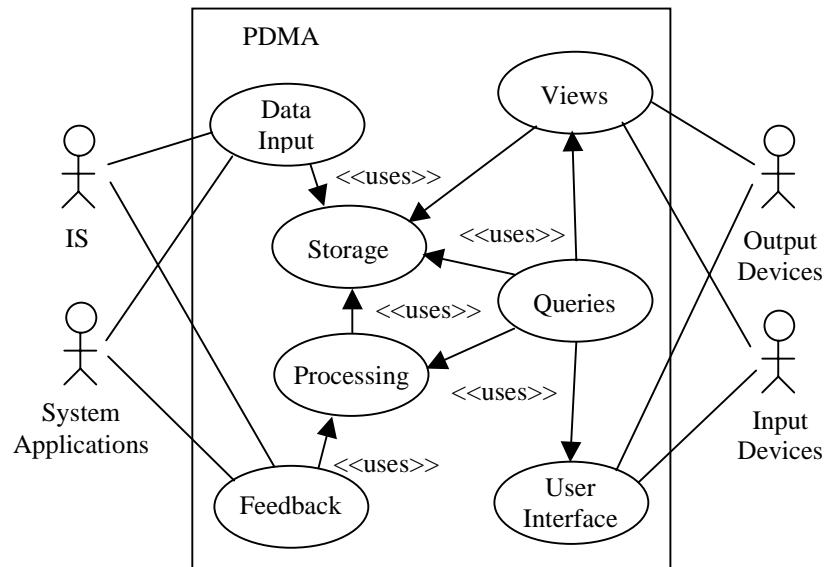


Fig. 1. A Use Case diagram showing the use cases, actors, and protocols for the PDMA. In this diagram, actors on the system represent classes of outside actors, and not individual items. The <<uses>> statement indicates that a use case uses the functionality of another use case (the end with the arrow)

A UML Class Diagram shows the classes that implement the architecture design and provides a vehicle for describing the details of the PDMA. The PDMA consists of three primary classes, System Interface, Analysis Context, and Data Warehouse. These primary classes are separated from each other to preserve data hiding principles and promote independence among system threads. These two principles provide flexibility for many specific system implementations [7].

The System Interface class (shown in Fig. 2) responds to large numbers of data inputs with short processing routines. Data inputs include performance events and statistics from the IS and configuration and loading data from system applications. This demands a relatively high priority thread to prevent queue overruns. It accepts data, converts to internal format as necessary, and routes the data. These items must be done quickly to prevent stealing too much time from other system threads.

It is also responsible for the output of data to the IS and systems applications. ISs accept feedback during operation to control the amount of instrumentation collected from specific instrumentation points. In this case, the data is formatted for output and routed to the IS using the appropriate interface. System applications can also accept feedback to control message routing and priority, system thread priority, and other features that may be determined by current and future research.

Flexibility is required to support different input/output sources. Local file access, shared memory, and object brokering from/to other nodes and applications are supported by the classes defined. The classes do not require specific object broker protocol, but provide an interface for object brokers. Existing object brokers can be utilized when the system platform allows for such. However, systems like embedded high performance systems often require custom solutions for speed and platform. The classes provided support this environment with interfaces designed for this task.

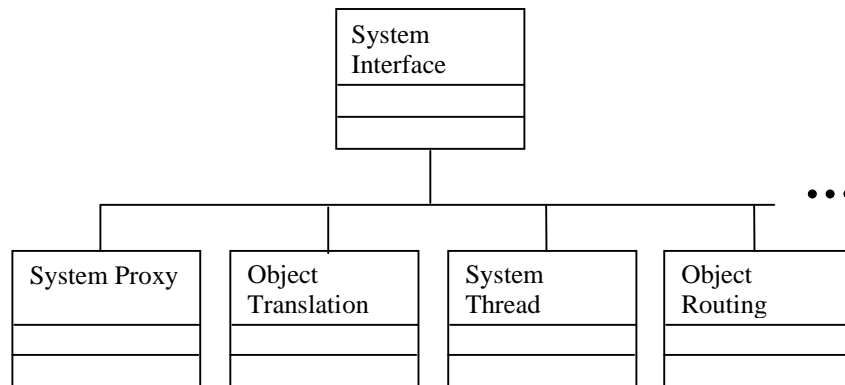


Fig. 2. Class Diagram showing the System Interface primary class

The Data Warehouse class (shown in Fig. 3) handles persistent data and responds to data storage requests and data search requests. It is also responsible for agents or database request to control size, important for embedded solutions with fixed memory. Data storage requests may include the computation of relational information that is stored with performance data. This class is separated to allow the use of a custom database structure or an off-the-shelf database application, which is dictated by the specific application. The processing priority of this task is likely to be low and require more time than the other classes, due to the nature of search requests, which is promoted by separation from the other classes. The interface to this class is tightly controlled through data storage requests and data query requests, enabling the update on either side of the interface without affecting the other.

An important factor for flexibility of the data warehouse implementation is relational information. The interface supports relational data requests and the formation of new relational information. Some key techniques for data mining include the search for new association rules, clustering, classification, sequential patterns, and outlier

detection [8]. These techniques are supported in this design, including the use of relational information. The combined use of system application data and performance data also provides new analysis possibilities for environmental tolerance and corrective action.

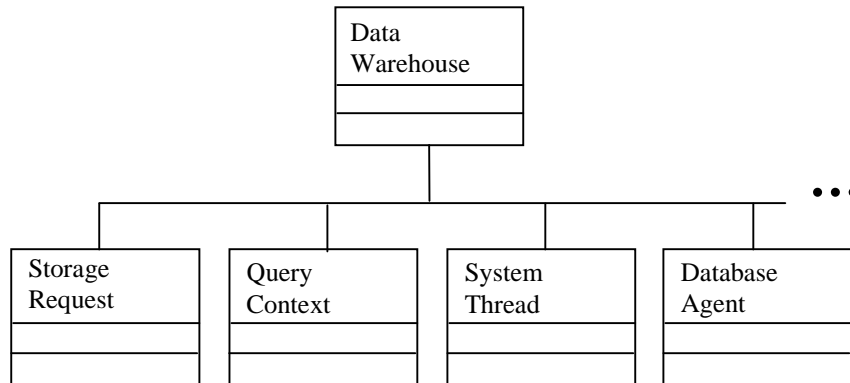


Fig. 3. Class Diagram showing the Data Storage primary class

The Analysis Context class (shown in Fig. 4) contains classes used for a specific analysis problem or context. Each performance analysis request has a specific context and can best be addressed by using an analysis context class instantiation, often running in its own thread. Allowing a separate thread for analysis contexts provides flexibility for assignment and priority of the thread. This supports a wide range of system applications. It is especially important to embedded high performance systems, where changing environmental conditions can be accounted for with dynamic adjustment of analysis threads.

Secondary classes under the Analysis Context class include classes for algorithms, display constructs, interfaces and translations to display hardware that is not high resolution CRT, contextual (display) and operator entered query formation, user interface, and others. The criterion for the definition of these classes is to allow the addition of new algorithm, view, and other objects without affecting the existing objects. Further, the specific system implementation, including hardware and software, must not affect the underlying PDMA, only a few classes defining the interface to such items as hardware or system applications.

The method for separating these analysis context objects is the interface to each of the objects. The Algorithm class can have many possibilities for computation within the instantiated object, but the interface to the View class, the Query class, etc., is maintained. A View object can then be utilized to display computed data, formatting the computed data in anonymous methods (from the algorithm viewpoint). The Display class receives View object data in a standard interface and transforms or translates the data to the specific hardware device involved in the Analysis Context instantiation. This may involve high resolution CRTs, character screen displays, banks of LEDs, alarms, etc.

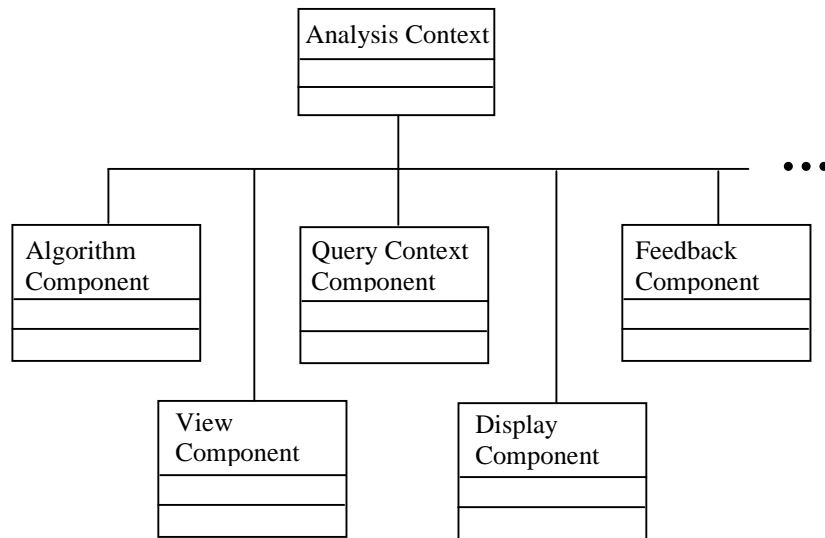


Fig. 4. Class Diagram showing the Analysis Context primary class

The Feedback class performs a similar function to the Display class, as it accepts the outputs from the View class, but it transforms or translates the data for feedback to the IS or system applications. It is separated from the View class because the necessary interface for each is unique enough to warrant it. This is shown by the types of data required by Display objects and Feedback objects.

The previous figures did not show the relationships between the classes. Relationships exist in the form of data objects, including performance data objects, query request objects, query request objects, view data objects, etc. Two of these these relationships are shown in Fig. 5. These objects determine a large part of the interface between the classes. Several more relations have not been shown.

4 Discussion

Current tools handle the presentation of data by providing several displays of data, such as Gantt, histogram, and pie charts, which can be selected. Some tools allow the user to select the data types to be displayed. In the PDMA, this capability is extended in an object-oriented method. A Gantt chart object is a class containing basic parameters such as data orientation, scale, etc. Instantiating a Gantt Chart object accepts the interface parameters and builds a display view (within the objects scope). The internal view of this Gantt Chart is not what is presented to the user however.

Additional modules within the display interface take the display parameters and map them to the display hardware. The display hardware will not always be a high

resolution CRT, the common display hardware in the lab. Embedded high performance systems may utilize character displays, banks of LEDs, klaxons, or some other hardware device. The display class handles this responsibility and allows the use of any views with any display technology.

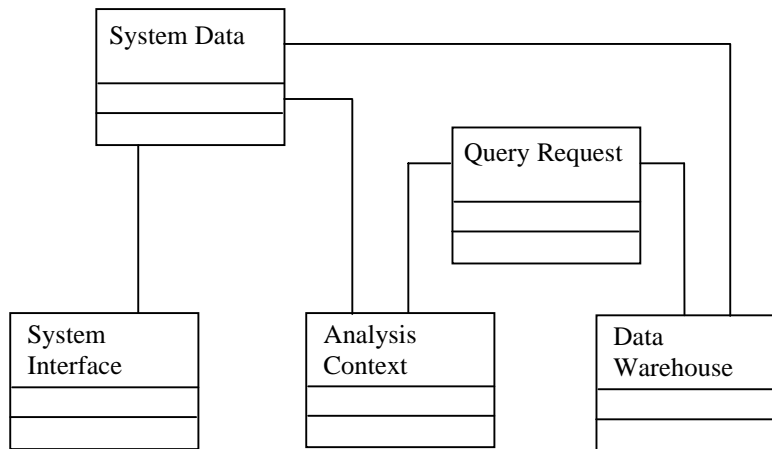


Fig. 5. Class Diagram for the PDMA showing two of the relationships between the classes

Paragraph and other tools limit the user to the predefined selections, since the system does not provide user definition of views, and the source cannot be easily modified. Using the PDMA, a user sets up an analysis context, including an instantiation of the desired view, scaling, data to be displayed in the view and its orientation, etc. Given this interface, the user can define these during operation. Further, the user can define new objects for views, etc., during operation with the user interface. The analysis context can also be instantiated as a performance monitor. In this case, no display is instantiated until an event of interest appears, at which point the display is created.

Additionally, priorities can be assigned to the context, and actions assigned to its results as well. The user can assign priorities or interface to a scheduling algorithm to control the scheduling of tasks to meet the requirements during any specific operating environment. Embedded high performance systems have complex operational environments, which are difficult to accurately predict and design for. Providing capabilities for the operator, coupled with system support, provides a more flexible environment and greater operational success.

The displays allow interactive queries, such as entered queries or button clicks in the context of a display. Each of the query types resolves the display context for mouse clicks, or resolves the textual entry of a query. This provides the interface to the Data Warehouse class, maintaining a simple constant interface to the database.

5 Future PDMA Research

This paper presents the definition of a PDMA considering a wide range of systems from a general point of view. It is purposely designed to promote future analysis research into view and algorithm technology, while allowing that technology to be readily exploited. Research on views, algorithms, data relationships, etc, are expected.

6 Conclusions

A Performance Data Mining Architecture (PDMA) has been presented, that objectifies and extends current tools, directly impacting embedded high performance systems. The design of the PDMA matches the design language of other systems, allowing the PDMA to be readily integrated. The PDMA provides support and interfaces for objects such as views and algorithms that don't require redesign of the PDMA. Finally the PDMA allows for portability because it is not dependent on a specific instrumentation system, or a specific operating system, or the hardware and software limitations of a fielded system.

Acknowledgements

This work was funded in part by DARPA Contract No. DABT63-95-C-0072 and NSG Grant No. ASC-9624149.

References

1. D. Reed et al., "Virtual Reality and Parallel Systems Performance Analysis", *IEEE Computer*, pp. 57-67, November 1995.
2. M. Heath and J. Etheridge, "Visualizing the Performance of Parallel Programs", *IEEE Software*, 8(5), September 1991, pp. 29-39.
3. D. Bhatt, et al., "SPI: An Instrumentation Development Environment for Parallel/Distributed Systems", *Proceedings of the 9th International Parallel Processing Symposium*, April 1995.
4. R. Agarwal, P. De, and A. Sinha, "Comprehending Object and Process Models: An Empirical Study", *IEEE Transactions of Software Engineering*, Vol. 25, No. 4, July 1999, pp. 541-544.
5. UML Documentation [Online], available at <http://www.rational.com/uml/>, April 30, 1999.
6. B. P. Douglass, *Real-Time UML : Developing Efficient Objects for Embedded Systems*, Addison Wesley Longman, Inc., 1998.
7. L. Bass, P. Clements, and R. Kazman, *Software Architecture In Practice*, Addison Wesley Longman Inc., 1998.
8. A. Zomaya, T. El-Ghazawi, and O. Frieder, "Parallel and Distributing Computing for Data Mining", *IEEE Concurrency*, Vol. 7, No. 4, October 1999, pp. 11-13.