Automatic Detection of Memory Anti-Patterns

Adriana E. Chis

Performance Engineering Laboratory
University College Dublin, Ireland
adriana.chis@ucd.ie

Abstract

In large distributed enterprise systems detection of memory problems can be a burdensome task. For understanding the memory related problems a catalog which documents memory anti-patterns is proposed. This paper also introduces our proposed prototype tool for the automatic detection of memory anti-patterns.

Categories and Subject Descriptors D.2.5 [**Software Engineering**]: Testing and Debugging – Debugging aids, Diagnostics, Tracing; D.2.8 [**Software Engineering**]: Metrics – Performance measures

General Terms Measurement, Documentation, Performance, Reliability

Keywords memory anti-patterns; memory footprint; Java heap dumps; metrics

1. Motivation

Distributed enterprise applications are becoming increasingly complex and problem determination during both testing and in production environments can be a difficult task. For example, enterprise applications can typically contain millions of objects and thousands of different object types. In such systems it is difficult to pinpoint the root cause of memory issues when they occur. These memory problems can have a negative effect on runtime performance and in severe cases can cause the application to crash.

Java heap dumps contain large amounts of data which can be complex and difficult to understand. Commonly, memory analyses require expert knowledge which is often not readily available throughout development and testing teams. Furthermore today's tools are limited in terms of the functionality they provide and the volume of data that they produce is large. The identification of the problems using current tools represents a burdensome task because a significant amount of time has to be spent on analyzing the data they produce.

OOPSLA'08, October 19-23, 2008, Nashville, Tennessee, USA.
ACM 978-1-60558-220-7/08/10.

2. Background and Related Work

For the automatic detection of recurring memory related problems the issues need to be first documented. Recurring issues in software are often documented in form of the anti-patterns. Brown et al. [4] define anti-patterns as "a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences". Anti-patterns provide understanding for what was wrong with the application and a solution that can be applied to eradicate the problem.

The automatic detection of the anti-patterns has been previously performed for performance design and deployment anti-patterns in component based systems [3]. Detection is achieved through a three stage approach: (1) data is collected from the systems under test, (2) analysis is performed on the collected data and a model is extracted, and (3) known problems are detected in the model.

To identify memory issues in the heap, the structure of the heap has to be known. A previous effort in this area has been made by Mitchell [2] who reveals the content of the heap through ownership analysis. Mitchell and Sevitsky [1] also provide two classifications for memory health where bytes are categorized by the role they have in objects (e.g. primitive, header, pointer, and null) and objects are categorized by the role they have in a collection (e.g. head, array, entry, and contained). Furthermore, they propose a technique for analyzing the individual data types and exposing their essential structure. This summarization provides a useful insight into memory consumption which can assist with bug detection. However analysis of such structures for memory issues requires expert knowledge. Our work provides automatic analysis of such structures which allows for automatic recognition of memory related problems.

3. Proposed Solution

The automatic detection of memory anti-patterns requires both documentation and a detection process. As part of this work *we are documenting a new catalog of memory anti-*

patterns. Furthermore *we are applying anti-pattern detection to the new area of memory related anti-patterns*.

3.1 Memory Anti-Patterns

A catalog that documents memory related issues is in development. It documents a large range of memory problems from the data structures that produce high overhead memory footprint [1] to more acute problems like memory leaks. The memory related anti-patterns are documented at two levels of understanding: for those who are not experts in the memory area (e.g. system testers trying to identify issues) and for those who have a deep technical expertise (e.g. experienced developers expected to resolve such problems). The problems are also documented from the detection perspective such that future tool developers can easily detect such issues. The data required for automatically detecting the anti-patterns will thus be documented.

3.2 Prototype tool for memory anti-patterns detection

As part of this work we propose a tool that automatically identifies the memory related problems from large distributed Java applications. The aim of this tool is to identify issues and to provide a precise description of the problem and a corresponding solution in a format that is useful for both systems testers and domain experts. This tool will provide support for answering such questions as:

- Is there a memory leak? If so, what is the source of it?
- Which data structures have the greatest memory overhead? Are there more efficient solutions?
- Does the heap contain data structure dependencies which are known to be inefficient? [1]

Our prototype applies two main steps for anti-pattern detection: analysis and detection.

3.2.1 Analysis

The analysis phase uses as input Java heap dumps. A heap dumps represent a complete snapshot of all live objects in the Java heap. Heap dump analysis retrieves the memory map of the application at certain point(s) during its execution. A number of different heap analysis techniques (such as object ownership, backbone equivalence) [2, 1] are used during the analysis step to extract a model of the heap.

3.2.2 Detection and Presentation

Documented anti-patterns will be transformed into metrics/ rules. The automatic identification of the memory anti-patterns will be achieved using the model from the analysis phase combined with static analysis and metrics. The model computed in the analysis phase is annotated with metrics. For example, for detection of a known anti-pattern, whereby large numbers of high overhead collections with few elements exist [1], three metrics can be used for detection: (1) the number of collection's instances, (2) the number of collection's entries, and (3) the ratio between the fix memory overhead (per container) and the variable memory overhead (per entry). If such metrics violate particular threshold values the anti-pattern is detected.

Visualization also represents an important part of this work. The heap content is displayed in UML-like diagrams as trees of data structures. For each data structure a zoom in option is available to look into composite objects. In this stage identified anti-patterns are highlighted and presented in a meaningful format, accompanied by the solutions.

4. Conclusion

There are two main outputs of this work. A catalog which documents memory anti-patterns is proposed. It will be presented to provide understanding to both experts and non-experts in the area of memory management. This catalog is useful from a problem solving point of view: the problem is explained in detail and a solution is suggested. Furthermore it will provide data required for automatic detection of the issues. Such a catalog will allow for the sharing of knowledge across the industry and for the development of tools for detection. The second output of this research will be a prototype tool for memory anti-pattern detection in Java heap dumps. The tool will present memory issues in a useful format for both system testers and domain experts. Another important aspect is that this tool will suggest a solution, which may target the user code, the libraries, or JVM itself. The tool will be validated by applying it to real enterprise applications from IBM.

Acknowledgments

Thanks to Nick Mitchell, Gary Sevitsky and Trevor Parsons for their useful comments and discussions. This work is funded under the Enterprise Ireland Innovation Partnership in cooperation with IBM and UCD.

References

[1] N. Mitchell and G. Sevitsky. The Causes of Bloat, The Limits of Health. In Object-Oriented Programming, Systems, Languages, and Applications, pp. 245-260, 2007.

[2] N. Mitchell. The Runtime Structure of Object Ownership. In European Conference on Object-Oriented Programming, pp. 74-98, 2006.

[3] T. Parsons. Automatic Detection of Performance Design and Deployment Antipatterns in Component Based Enterprise Systems, PhD Thesis, University College Dublin, 2007.

[4] W. J. Brown, R. C. Malveau, and T. J. Mowbray. AntiPatterns: Refactoring Software, Architectures and Projects in Crisis. Willey, 1998.