

Autonomic computing in Canadian academia, Part 1

Autonomic computing projects in the Toronto Centre for Advanced Studies

Skill Level: Intermediate

[Elizabeth Dancy \(lizdancy@ca.ibm.com\)](mailto:lizdancy@ca.ibm.com)

Software Developer

IBM

[Uliyana Markova \(umarkova@ca.ibm.com\)](mailto:umarkova@ca.ibm.com)

Development Project Manager

IBM

15 Apr 2008

As IBM® grows and develops autonomic technologies, the autonomic computing initiative relies heavily on research and development to present innovative ideas, improve existing technologies, and prototype in the most rapidly expanding development areas. In this article, take a look at two autonomic computing academic projects that are bringing new developments to IBM. The first project looks at converting legacy code to source code that is autonomic-ready, while the second project focuses on new methods for identifying problems in large-scale applications. For each project, you will understand the current research direction and then explore the project in detail. Finally, you will see how each project contributes to the Monitoring-Analysis-Planning-Execution (MAPE) loop design model and what future research directions are planned.

Introduction

Many of the problems and questions faced by IBM development teams are first explored as research projects in conjunction with top Canadian Universities at the

IBM Centre for Advanced Studies (CAS). In Toronto, Ottawa, and Montreal, the CAS centres serve as incubators for the latest ideas, with a goal of producing new innovations and solutions relevant to IBM products as well as strengthening ties between IBM and academic CAS collaborators.

As the IBM autonomic vision materializes, a host of new research problems are created, many of which are specific to an IBM development goal or product. Therefore, CAS is the ideal environment to engage top students and professors to conceptualize real, usable, deliverable solutions that can then be integrated into IBM products. This is not a trivial feat. CAS students, professors, and collaborators must cooperate to produce answers to IBM's most pressing research questions while maintaining conformance with the original IBM autonomic vision. Each respective solution must provide services that are compatible with the overall model of autonomic computing. This article gives you an understanding of the following two CAS projects:

- **Software Tuning Panels for Autonomic Control (STAC) II:** a project that aims to automatically identify source code tuning parameters that can then be used in a re-architecture of the source code to facilitate autonomic control.
- **Problem Determination in Enterprise Software Systems:** a project that focuses on using anomaly detection to identify problems in large software systems at an early stage.

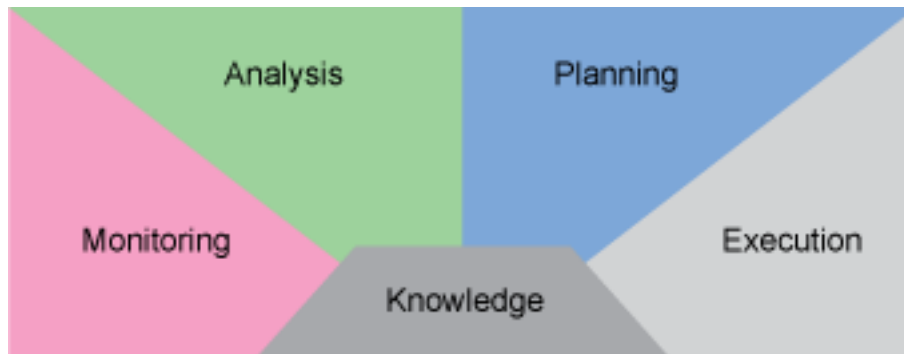
This article first outlines the overall goals and progress to date for each project and then shows where these projects fit into the autonomic computing architecture. You will also gain knowledge about the future direction of each project.

The MAPE loop

Early on, IBM identified the MAPE loop, shown in Figure 1, as the gold standard architectural model for autonomic computing. In the MAPE cycle, an autonomic controller performs activities to accomplish each of the four phases represented by the four quadrants: Monitoring, Analysis, Planning and Execution (for more information, see *An architectural blueprint for autonomic computing* in [Resources](#)).

By following this process, an autonomic manager can use shared knowledge to perform autonomic functions and enable the resource to be self-controlling.

Figure 1. MAPE loop



Often, an autonomic product or service does not readily display how it fits in the MAPE loop, but getting the big picture is increasingly important as you look toward high-level integration of numerous autonomic products. For this reason, this article includes a brief explanation of how each featured project contributes to the MAPE model.

Project: Software Tuning Panels for Autonomic Control

Research problem

When working on a large initiative such as autonomic computing, it is tempting to focus your attention on constructing novel plans, models, and best practices. Yet it is impractical to overlook all of the existing functions already implemented without regard to becoming autonomic-ready. If the autonomic vision is to be realized, the large supply of legacy source code already available must be addressed and adapted to embrace autonomic technologies. It is this idea of converting what you already have to become autonomic-ready that forms the motivation for the project Software Tuning Panels for Autonomic Control (STAC). The high-level goal of STAC is to provide an end-to-end process where source code tuning parameters can be automatically identified and exposed in a control panel module serving as a central point of control for an autonomic controller.

The control panel is important to autonomic computing because having the ability to self-control and adapt to different situations (that is, to become autonomic) relies on tuning or modifying certain parameters over time as the code runs or as a response to some situation. The challenge of re-architecting is difficult because many well-functioning legacy systems scatter their tuning parameters in the source code in such a way that the parameters do not have direct accessor or mutator methods.

Therefore, accessing them for modification requires a deep level of program understanding to avoid causing a chain reaction of dependency breakage. It's important to have access to the hidden parameters because they can act as the control knobs for an autonomic controller to facilitate self-monitoring and self-tuning.

In 2006, a proof-of-concept project called STAC I was completed. This project

demonstrated the use of a small prototype tool showing the automated re-architecture of source code to expose specific tuning parameters for an autonomic controller. The major limitation of STAC I was that it required the tuning parameters it would use for re-architecture to be manually identified by a human user prior to the transformation. STAC I then proved that the code could be successfully re-architected, but did not address the problem of autonomically determining the tuning parameters.

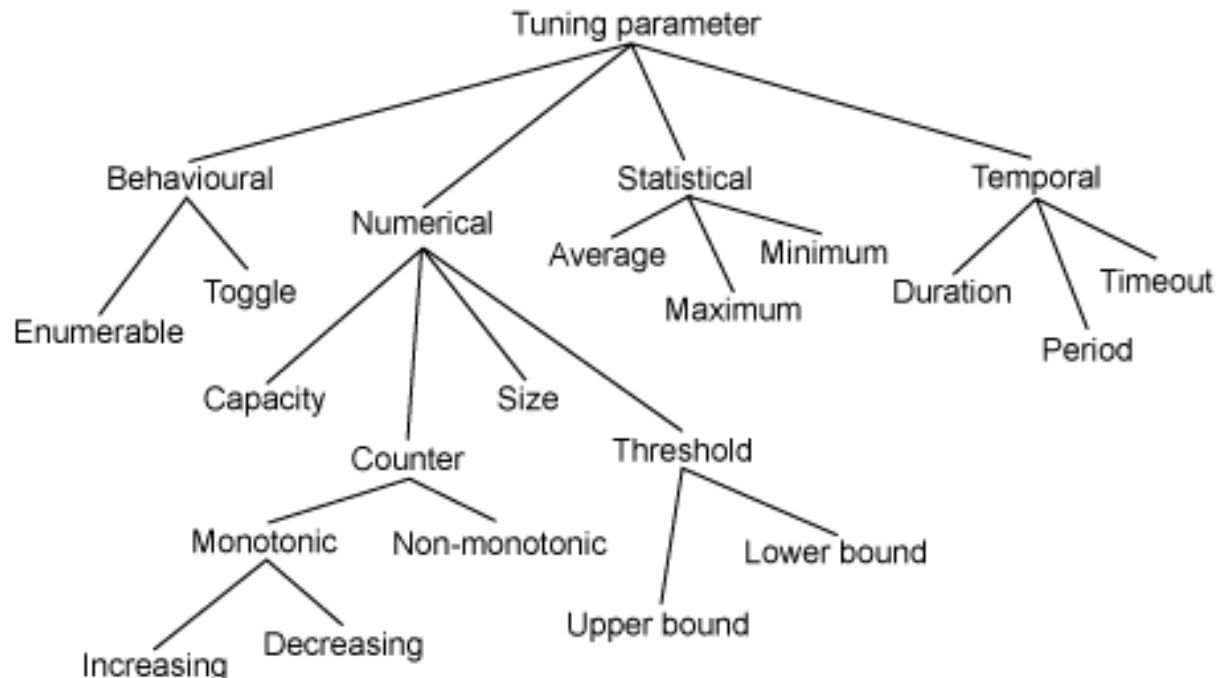
Not only is manual identification time consuming, it also requires a great deal of program understanding and parameter classification that is suitable for automation. STAC II (a subset of STAC) is an IBM CAS-based research project, in conjunction with Queen's University, that is focused on automating the process of finding interesting tuning parameters, the most complex piece of the problem addressed by STAC, for use with the STAC I prototype. The remainder of this section describes the recent progress of the STAC II project and its applications.

Current scope and research progress

STAC II has progressed over the past year from a high-level process design to an implemented solution. To automate the identification and classification of tuning parameters given only the source code and the documentation, the first step of designing STAC II was to precisely define what is to be searched for across all applications; that is, to perform an empirical study of existing software systems with documented tuning parameters and then classify the results into known tuning parameter categories using a taxonomy. Industrial-sized applications were used as the basis for the classification, including Apache Tomcat and Apache Derby.

The taxonomy created is based on usage patterns within the source code and is shown in Figure 2 (from "STAC: Automatically Identifying Software Tuning Parameters"; see [Resources](#)). The entity-relationship model was built to represent source code elements such as types, fields, local variables, and methods, and the relationships between them. For example, a code pattern for variables that keep track of something is defined as a *counter*. Using fact-extraction techniques as well as manipulation of graphs constructed from known tuning parameters lets documented parameters be identified in the source code (finding a previously documented parameter through pattern matching) as well as new parameters to be identified (identification of a previously undocumented parameter).

Figure 2. Tuning parameter taxonomy



The functionality was built into an Eclipse-based tool that searches the source code for the patterns identified through these methods. For example, when given as input the source code for a small client-server messaging application, STAC II may identify in its list of parameters a variable, `userCount`, which represents the current number of active connections to the messaging server. STAC II also identifies that this variable falls into the counter category as defined by the taxonomy in Figure 2 because it counts the number of active connections. The variable may be useful for an autonomic controller to know about while monitoring the messaging server, perhaps so that some action may be taken when a certain number of connections is reached. But, there may be dozens of references to `userCount` in the source code, and monitoring all of them would be futile. After STAC II has identified the parameter, STAC I can go to work to expose it in the control panel so that the autonomic controller can directly monitor it by attaching only to the control panel and paying no attention to the dozens of references to `userCount` in the source. This completes the goals of the STAC project.

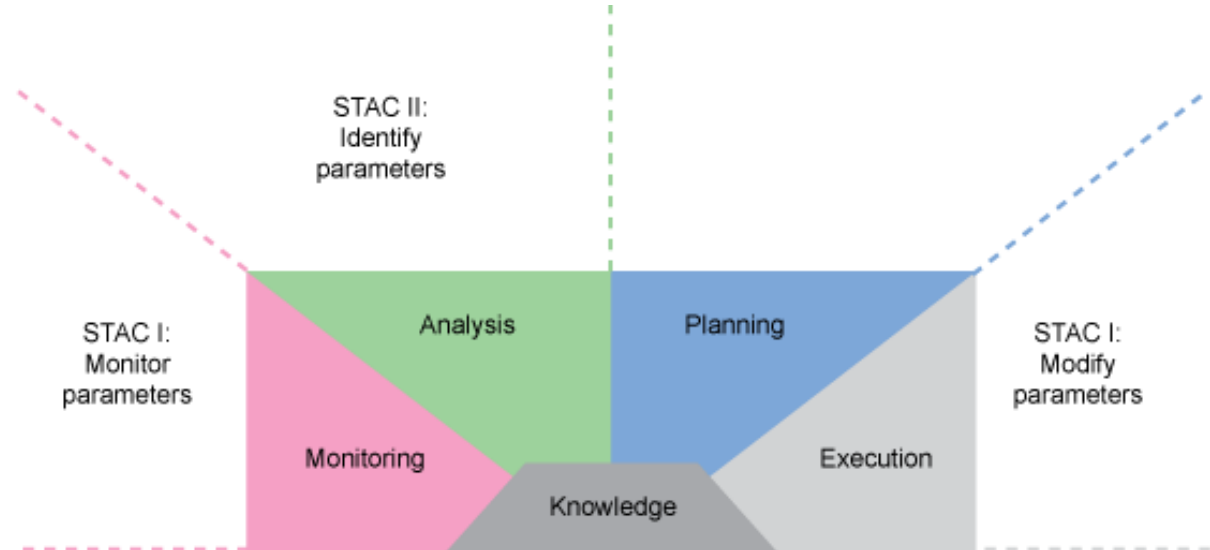
MAPE loop highlights

The STAC II project demonstrates the analysis phase of the MAPE loop as shown in Figure 3. If you imagine that the application is the managed resource, an autonomic controller can monitor the source code through the control panel created by STAC I after the STAC II tool has performed analysis on the source and provided a list of tuning parameters. The tuning parameters may or may not have been documented. However, this tool is most useful in cases where you have little or no documentation of tuning parameters, which in reality is not uncommon in legacy code.

STAC I can be used in the monitor phase of the MAPE loop to monitor for parameter value changes (as in the `userCount` example) and then feed the monitoring data into the planning phase to make decisions on how to self-tune. Alternatively, it can be used in the execution phase of the MAPE loop, allowing the autonomic controller to modify the tuning parameters in the control panel at run time.

For example, if you have a tuning parameter that tracks the maximum number of connections to a server, STAC will provide you with a way to access this variable to see whether there are too many connections at a certain time. This demonstrates the monitoring capabilities of STAC. As a second example, imagine you have a threshold variable that is used to decide when there are too many connections. At run time, an autonomic controller may change the value of this threshold based on monitoring data it receives from another parameter. This demonstrates the tuning capabilities of STAC. Figure 3 shows the MAPE loop elements addressed by the STAC project.

Figure 3. STAC in the MAPE loop



Integration goals and natural extensions

The work done in STAC II presents many useful applications. Most obviously, it is an invaluable development tool for programmers who are focused on transitioning legacy source code to be managed by an autonomic controller. Furthermore, the tool can aid in understanding the program and automating the creation of documentation for the legacy code, key advantages in code maintenance when the original programmers of legacy programs are longer around and there is no documentation.

STAC II can also be used in conjunction with STAC I for powerful tuning of applications. Because the tuneable parameters are housed in their own separate module, the tuning can be done dynamically at run time or statically after analysis. This enables black-box tuning, removing the prerequisite level of source code understanding usually required to make tuning parameter modifications and allowing

this to be readily automated.

The future goals for STAC II include:

- The validation of the patterns against a variety of systems that it was not specifically developed from. The categories were created from an original empirical study of software systems and the validation systems will be outside of the original system set.
- The end-to-end application of the STAC tooling in a product-level code package. The STAC II tool presents a good avenue for testing the capabilities of STAC II in various domains to determine which product or set of products will benefit the most from this innovative new solution.

Research problem

Enterprises continue to spend a large amount of time and human resource on maintaining the smooth running of their software systems. Despite this effort, determining the cause of a problem within the system and finding the solution for it is often a lengthy and sometimes unsuccessful process. In production systems, such as online store systems, banking applications, or travel booking systems, these delays can be costly both in profit and in customer satisfaction.

IBM, in conjunction with the University of Waterloo through the Center for Advanced Studies, is driving a research project for problem determination in enterprise systems that is based on a refreshingly new idea. Simply put, the idea is to build a statistical model of a healthy system and during operation, apply this model to the event data collection infrastructure to determine discrepancies between the running system and its healthy version, and notify the system administrator of the problem occurrence. As a second benefit, if this statistical model is applied to the monitoring metrics collected by the monitoring system of choice, a much smaller subset of what should be monitored can be identified. This will result a cost reduction for continuous monitoring.

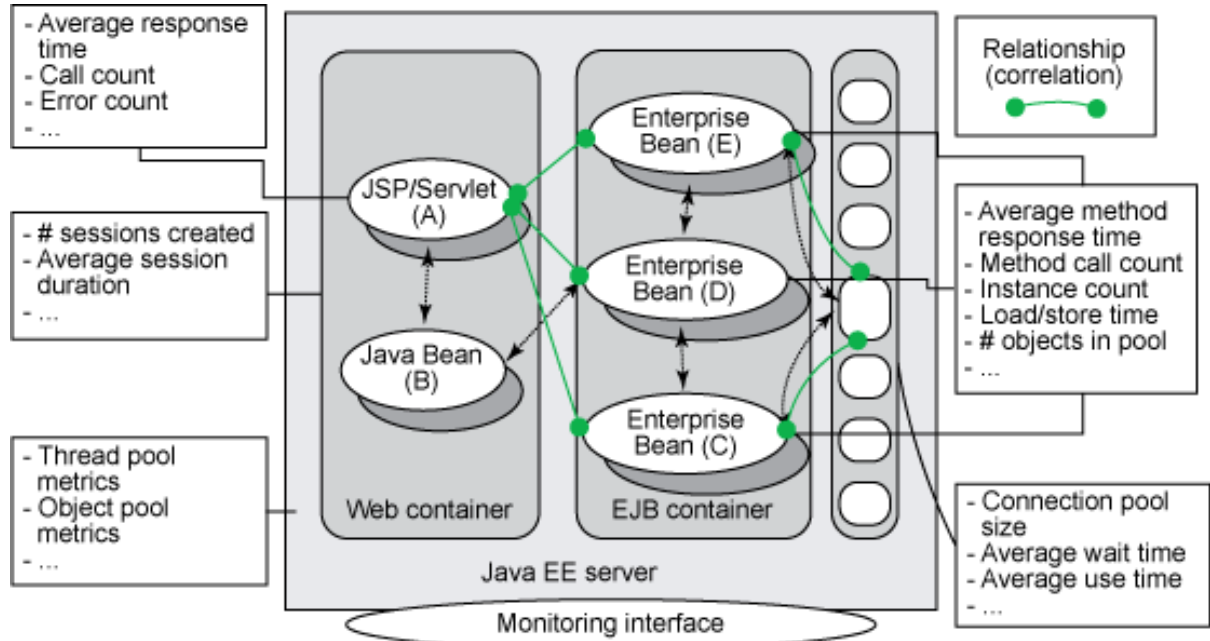
Current scope and research progress

The group of researchers has identified a method for building a statistical model of a normal system's behavior by collecting a large set of monitoring data during normal operation over a period of approximately 15 minutes to half an hour. The monitoring data is considered to be two main types: event data such as what is recorded in log files and traces, and management metrics, such as Java™ Management Extensions (JMX), Windows™ Management Instrumentation (WMI), and the IBM Tivoli® monitoring data.

How was the statistical model of metrics for a normal system created? First, the researchers looked at the monitored metrics as individual entities. As you may have observed in a CPU consumption monitoring window, the metrics typically behave

non-linearly in time. This presented difficulty in analyzing the metrics. The next idea was then to look at metrics in relation to each other. First, a number of metrics were identified in relation to each of the selected system entities. Such entities in a Java Platform, Enterprise Edition (Java EE) server are Java beans, JavaServer Pages (JSPs), and servlets. Next, relationships were identified through pairs of related metrics as shown in Figure 4 [1].

Figure 4. Relationship invariants



[1] Mohammad A. Munawar, Thomas Reidemeister, Michael Jiang, Paul A. S. Ward, Eric Labadie, Marin Litoiu "Problem Determination in Enterprise Software Systems" [Poster]. *CASCON 2007 Technology Showcase*, Toronto, October 2007.

An example of such a pair is: Average JDBC Operation Time and Average Entity Bean Method Response Time. The green arrows in Figure 4 represent relationships between entities with correlated metrics. The correlations established within these pairs proved to be of linear character and were easier to track. In the pair example, the Average Entity Bean Method Response Time increased when the Average JDBC Operation Time increased.

The "training" of the statistical model included preservation of the correlations that were observed throughout and removing the ones that did not hold until a finalized set of correlations was established. This set of correlations is the statistical model of metrics for the normal system. This set of correlations could then be compared to that of the currently observed system to identify anomalies and, thus, potential failed operations for the entities associated with them.

This approach to monitoring is used to identify a small subset of management metrics to be monitored when conditions are normal and invoke a detailed

monitoring for the cluster of correlated metrics when a failure occurs. This results in low cost monitoring and, then when a potential failure is observed, on demand monitoring modification. This method fits in the monitoring phase in the MAPE loop shown in Figure 1.

How is this approach employed for monitoring event data? The same idea of comparing healthy data to current data is applied with the goal of raising the administrator's attention only to event data that is potentially linked to faulty situations. How is this done? The Log Analyzer tool provides the capability to visualize the events read from the log files in a selected time window. The tool was also given the ability to create histograms from selected events. When an error is injected using the IBM Resiliency Benchmark tool, a histogram of events is created for the time window before the error (normal histogram) and after the error (observed histogram). The visual comparison of the bar chart graphics representing the histograms of events from a normal and observed system behavior allows for immediate identification of the new event or events that occurred in the time when the error was active. As a result, the newly identified events are now linked to the error that was injected.

This mapping is used to build a behavioral model of the system and identify symptoms for common faults. This method fits in the planning phase in the MAPE loop shown in Figure 1. The model when applied to a system in production will cause the system to generate Common Base Events whenever there is a statistically significant deviation in the behavior of the system from that predicted by the model, which can be further analyzed using the Log Analyzer. This analysis fits in the analysis phase in the MAPE loop.

Future work

The initial model was based on an experimental 3-tier setup including a workload generator (IBM Rational® Performance Tester), the IBM WebSphere® Application Server and the IBM DB2® UDB 8.2 database system. To bring this approach to market, the next step is to integrate with the IBM Tivoli Monitoring (ITM) product so that the method can monitor a much larger set of software components, including operating systems, databases, and HTTP servers.

Future work is planned for automation of both symptom generation, based on statistically significant symptoms, and system recovery, based on the event analysis engine used in the Log Analyzer tool. In addition, improving the robustness of the model, adding continuous data analysis for log data, and extending the range of faults that can be detected and diagnosed are other goals that this project will address.

Conclusion

The two CAS projects examined provide you with a small window into the current

autonomic computing research being done in the IBM Toronto Centre for Advanced Studies. You learned first about efforts to facilitate autonomic control in legacy source code as examined by the STAC project. Next, you learned about data minimizing methods for identifying problems in large-scale applications by studying the Problem Determination in Enterprise Software Systems project. Both projects demonstrate innovative solutions to real problems faced by IBM developers who provide autonomic solutions for their consumers within the problem determination and monitoring domains. You have seen where each project can assist an autonomic controller in carrying out the phases of the MAPE loop, thereby highlighting their potential for re-use in autonomic systems. The second part of this series will allow you to explore two additional IBM CAS Toronto projects and provide a glimpse into the research and development problems being tackled in the area of resource provisioning.

Resources

- [Participate in the discussion forum for this content.](#)
- ["STAC: Software Tuning Panels For Autonomic Control"](#) E. Dancy and J.R. Cordy, *Proc. CASCON'06, 16th IBM Centre for Advanced Studies International Conference on Computer Science and Software Engineering*, Toronto, October 2006, pp. 146-160: This paper explains the STAC project.
- [IBM Centre For Advanced Studies](#): Learn about more studies being carried out at the IBM Centre for Advanced Studies.
- ["STAC: Automatically Identifying Software Tuning Parameters"](#) [Poster]: N. Brake and J.R. Codry . *CASCON 2007 Technology Showcase*, Toronto, October 2007
- [An architectural blueprint for autonomic computing](#): In this updated architectural blueprint, learn the latest advances in the architecture and standards as well as how the architecture can be applied in solutions that can achieve real business value.

About the authors

Elizabeth Dancy

Liz Dancy is software developer on the Autonomic Computing Log and Trace project. She has a special interest in adaptive systems and source code analysis. Liz holds a Master of Science degree in Computer Science from Queen's University.

Uliyana Markova

Uliyana Markova is a project manager of the Autonomic Computing Log and Trace assembly of common components. She has worked in the IBM Toronto Lab for 10 years, and as a project manager in the autonomic computing group for 3 years. Uliyana holds a Master of Science degree in Human-Computer Interaction from Heriot-Watt University in Edinburgh and a degree in Computer Science from Sofia University, Bulgaria.

Trademarks

The, IBM, the IBM logo, Tivoli, DB2, Rational and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries or both.