

# Towards Knowledge Management In Autonomic Systems

Thomas Cofino, Yurdaer Doganata, Youssef Drissi, Tong Fin, Lev Kozakov and Meir Laker

*IBM T. J. Watson Research Center*

*Hawthorne, NY 10562*

*{cofino,yurdaer,youssefd,thfin,kozakov,meir}@us.ibm.com*

## Abstract

*The classical definition of knowledge management promises to get the right knowledge to the right people at the right time so they can make the best decision [1]. Autonomic systems, on the other hand, are expected to find and apply the right knowledge for self-managing purposes without human intervention. This article discusses the components to be built around a system to enable self-healing and managing capabilities. These are defined and described in this article as self-knowledge, self-monitoring, self-learning, problem detection, diagnosis, search and solution application components. Interaction of these system components to make knowledge available for self-healing purposes will also be discussed.*

## 1. Introduction

The evolution of autonomic systems will start with the efforts to convert existing systems into self-healing and managing systems. These initial autonomic systems will learn self-healing adaptively and with human intervention at the same time. In these early stages of autonomic computing, it is very important to identify the footprints of an autonomic system by defining the components to be built around existing systems that will increase the self-healing ability. Since early years of 1980s, researchers have been dreaming about building trouble free computing systems [2], [3]. There are proposals for making systems more scalable, reliable and available [5], and attempts to shift the focus of research agendas from performance dominated research to recovery-oriented research [4] in the literature. Following the evolutionary steps of autonomic computing, this article is an attempt to identify the system components that will be essential to convert existing systems into self-healing and managing autonomic systems and build new systems with self-healing capability.

The concept of self-healing first requires automatic detection of failures or impending failures, without which it is impossible to take action. In order to be able to decide that a system has failed or is about to fail, one has to know what is “expected” from that system. Autonomic systems must know themselves [6] or must have a “self-knowledge”, which is used to determine if their current behavior is consistent or expected with respect to the environment. Under new conditions and the environment, new behavior may be observed. In this case, the “self-knowledge” requires an update. Hence, “self-knowledge” continuously evolves in time with the addition of new components, updates of the existing system components and with new usage scenarios. In other words, autonomic systems learn continuously about their own limitations and capabilities.

Functional specifications and requirements can be used as first steps towards building a “self-knowledge” for a system. Before a system or a product is released, the capability of the system to fulfill its design promises is checked by running stress tests. In the context of autonomic computing, self-identification tests can be run and the results can be used for building an initial “self-knowledge”. Today, service agent (SA) technology is used to collect information on the hardware and software components installed and fix patches applied in customer machines [7]. The SA technology could potentially be used in autonomic systems to increase the system’s “self-knowledge”.

In Figure 1, the components that are needed to make a system autonomic are presented. The system is driven by some test scenarios or tasks. The response of the system to those input scenarios is monitored by taking measurements at selected key points. This is very much like looking at the “impulse response” of a system to determine its characteristics. The “self-monitoring” system component collects the measurements and analyzes by taking into account the current environment. The results are then compared to the data obtained from the “self-knowledge” base. This is done to determine if the response of this system is “expected”. Major conclusions can be driven from this analysis. If the

system behavior contradicts the known behavior or “self-knowledge”, then this is an indication of a possible problem. If a new behavior is observed from possibly a new scenario, then the “self-learning” process is invoked which eventually updates the system’s “self-knowledge” base.

The designers of an autonomic system must determine the key points at which data should be collected to determine the health of the system. In a typical search system, for example, predetermined queries and their corresponding returned result lists along with the expected response times can be used as benchmark measurements to check if the system is alive - that is, functioning as expected. The designers of the system are the most knowledgeable people to best identify these key points for vitality signals. A self-monitoring process continuously probes for the data generated by these key system measurement points and analyzes the measurements to identify abnormal behavior.

The “self-knowledge” should at least contain the list of components that the system is made of such as software and hardware components and their capabilities. The “self-knowledge” may also contain set of rules, conditions and operational patterns expressed as functions of multiple inputs and corresponding outputs of the system. These rules are derived from the functional specifications or the design goal of the system. One possible way of describing these rules can be as follows: If the input is A, then the output is B when the condition is C. The self-monitoring process runs these rules to check if there is any inconsistency between the self-knowledge and the current behavior. If inconsistency is detected, control is transferred to another process for further analysis.

Detecting inconsistent behavior is relatively easier than formulating a problem statement that captures the inconsistent behavior. The main function of the problem detection process is to formulate the problem based on the data received from the “self-monitoring” system. The problem detection process determines how much deviation from the original design specifications and initial test results warrants classifying the observed behavior as a problem. The detection process should also take into account the current environment that is relevant to the system. There is a difference between detecting a problem and providing a diagnosis. Detecting a problem requires comparative analysis, while diagnosis requires identifying the nature of the cause of the problem.

In order to be able to come up with accurate diagnostics, advanced search systems and technical support knowledge bases are essential. The function of the problem detection process is to alert the autonomic system components that a problem exists. Formulating a query or describing the problem from the data that instigated the identification of a problem could also be

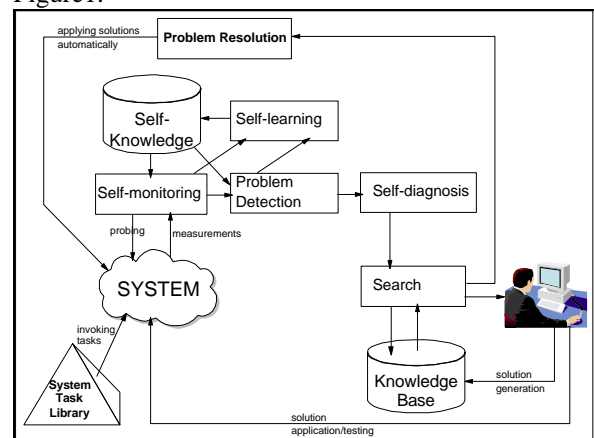
part of the problem detection process. Once the question or the problem statement is formulated properly, it is sent to the search system for a possible solution. For certain problem types, the search system must provide a unique solution and the steps required to apply that solution. Advanced question and answering systems, and case based or reason based problem determination technologies are candidate KM technologies for autonomic computing for their potential to provide solutions for technical problems.

The solution to a particular problem may differ from one system to another. Hence, the search systems to be used in autonomic systems must have the capability to take into account the specifics of a system or a product. As an example, IBM’s Premium Search utility for ITS customers uses data collected from customer machines to modify queries to deliver more personalized services. If certain fixes were already applied to a machine, the system uses this knowledge to provide a solution based on the fixes that have not been applied yet.

If the system cannot find a unique solution to a problem, hence applying the patch to fix the problem automatically is not possible, then human intervention may be necessary. As new problem resolutions are obtained by human intervention and new solutions are authored for autonomic systems for self-healing, the frequency of human intervention will be reduced.

In summary, the design and the development of autonomic systems will derive new requirements in the area of knowledge management. The traditional human centric methods of formulating queries, questions as well as the techniques to author solutions will change. The efforts to build effective self-healing systems will eventually drive a new field in the area of knowledge management, knowledge management for autonomic systems.

In the next section, the critical components of autonomic knowledge management and their interaction are presented. These are the components depicted in Figure 1.



**Figure 1. Autonomic system components**

## 2. Anatomy of autonomic knowledge management

### 2.1. Self-knowledge

The “self-knowledge” contains all the essential system information and the ability to represent this information as an explicit knowledge about system configuration, capabilities and operations. It is the system internal knowledge base. Some portion of the system information may be loaded into the “self-knowledge” module during initial system setup, while other information may be acquired during the system life cycle. Except for the basic static system attributes, like model number or system type (desktop, laptop, mobile, etc.), most of the information in the “self-knowledge” module may be updated as the system evolves.

The following categories of system information may be presented in the knowledge base of the “self-knowledge” module:

- System identification information
- Information pertaining to the design goals or mission of the system
- History of problems and applied solutions

Initially, the “self-knowledge” module is loaded with predefined system identification information. During the system lifetime, the “self-learning” module amends the system knowledge base by processing the information collected by the “self-monitoring” module. The “self-monitoring” module interacts with the “self-knowledge” module to perform comparative and statistical analysis of collected system performance data. The “problem detection” module also may interact with the “self-knowledge” module in order to identify a problem and find information about previously applied solutions.

### 2.2. System task library

All the system test scenarios are run by one or more specialized system tasks. We can distinguish the following categories of the specialized system tasks:

- Self-identification tasks
- Self-diagnostic tasks
- Self-monitoring tasks

The whole set of the specialized system tasks is included in the System Task Library. The tasks in the System Task Library are organized by categories of functions to be performed with specifications of the operational environment, as well as other necessary information. The System Task Library catalog allows performing effective task lookup, based on the predefined

task categorization. An example of system diagnostic tasks can be found in [19] where a system task library is provided for the purpose of performing diagnostic procedures in AIX systems.

### 2.3. Self-monitoring

The “self-monitoring” process collects data and takes measurements at selected key points. The collected data are used to compare the data reside in “self - knowledge” base to determine if the system behaves as expected. If the “self-monitoring” component detects an unusual behavior, it routes the corresponding data to the “problem detection” module.

The “self-monitoring” module probes and analyzes vital signals and data continuously, and runs tasks from the System Task Library. These tasks aim to capture errors and exceptions as well as collecting statistics and performance data. The “self-monitoring” module also sends the monitored data to the “self-learning” module, which runs learning processes, extracts new knowledge, and updates the “self-knowledge” component.

Building a “self-monitoring” system requires knowing what to monitor. This information is extracted partially from the “self-knowledge”. In an ideal “self-monitoring” schema, everything that is considered vital in the system should be monitored. Effective instrumentation, ideally non-intrusive, is needed to reduce or eliminate the extra overhead entailed by self-monitoring. The type and shape of the self-monitoring sensors depend on the different structural levels at which they operate. For instance, they can operate on the whole system, or on a particular layer, module, component, object, data structure or variable. Monitoring is also done at different functional levels to check the compliance of functions delivered by the system components with the expected behavior documented in the “self-knowledge”.

Event notification [8, 9] is a key concept in the process of enabling self-monitoring [10-18]. A “self-monitoring” system should be designed to generate meaningful and self-explanatory event notifications. Monitoring performance data and resource usage usually gives very helpful clues about the root cause of an error or exception. This type of data should be collected long enough to run statistical analyses. Time is an important parameter since variations during a certain time interval may need to be studied. Such performance and resource usage data include response times, CPU and memory usage, number and types of processes and threads, etc.

All the data monitored should be stored as well-formatted and self-explanatory logs. These logs need to be expressed in a structured language to allow their exploitation without human intervention.

## 2.4. Self-Learning

A computer program qualifies as a learning machine if its performance improves with experience. Experience is best understood as the knowledge gained from the analysis of several tasks performed during a period of time. In autonomic systems, the “self-learning” component is responsible for updating and improving the “self-knowledge” component. This is done through acquiring new knowledge learned from the data collected by the “self-monitoring” component. Knowledge about how a system should behave is essential to decide what went wrong with the system and in return what the system needs to learn in order to avoid a similar problem in the future. The data obtained through the “self-monitoring” module can be exploited by the “self-learning” component to discover and extract new knowledge from it. This extracted knowledge should enrich the “self-knowledge”, and therefore contribute to the improvement of the autonomic system’s performance as depicted in Figure 2 below.

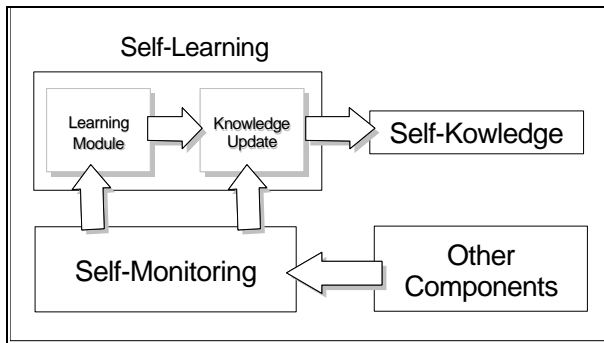


Figure 2. Self-learning & components interaction

## 2.5. Problem detection

The responsibility of the “problem detection” module is to analyze the input data collected by the “self-monitoring” module and establish the state of abnormal system response or failure. The input data may contain some of the following items:

- A detailed report of unexpected system response or state;
- A system or application event that describes a problem or failure that occurred under certain conditions;
- A detailed specification of the system environment at the moment when the data have been collected.

The “problem detection” module classifies the input data and creates an appropriate “*problem detection request*”, which includes all essential data and specifications. The purpose of creating a “problem detection” request is to find information that could help to

establish the state of abnormal system response or failure. Once the “problem detection” request is created, it is submitted to the “self-knowledge” module in order to find an appropriate *registered problem report*. The “self-knowledge” module may or may not contain the registered problem report that matches the current “problem detection” request. If an appropriate registered problem report was found, the “problem detection” module tries to retrieve the *problem solution record*, associated with the found registered problem report. Once the problem solution record is found, this record is passed to the “problem resolution” module to initiate the problem resolution cycle. If there is no complete problem solution record associated with the given registered problem report, the “problem detection” module creates the *problem diagnostic request* and submits it to the “self-diagnostic” module in order to collect more details related to the identified problem. Then, the “problem detection” module repeats an attempt to retrieve the necessary problem solution record, etc.

If the “self-knowledge” module does not contain an appropriate registered problem report, the “problem detection” module submits the “problem detection” request to the centralized knowledge base. If an appropriate registered problem report was found, this report is added to the “self-knowledge” module through the “self-learning” module, so the next time the same kind of problem occurs, the “problem detection” module will not need to search the centralized knowledge base.

Eventually, if an appropriate registered problem report was not found, the “problem detection” module creates and posts the *pending problem report*, if it was not already created. The pending problem report is posted to the centralized Knowledge Base to be handled by a human expert, who should create an appropriate registered problem report and solution record. Once the registered problem report is created, all interested systems are notified, and the suspended problem detection cycle can be resumed.

Figure 3 shows main interactions between the “problem detection” module and other elements of the autonomic system architecture.

## 2.6. Self-diagnosis

The role of the “self-diagnosis” component is to find the causes of the problem and to verify that the applied solution can fix the problem and the system then works properly. In the process cycle of determining the problem, the “problem detection” module may need to get detailed data and it will communicate with “self-diagnosis” to elaborate on the data. “Self-diagnosis” also interacts with “problem resolution” to ensure that the applied solution has fixed the problem. As surveyed in [4], there are many approaches to problem diagnosis.

The process cycle of “self-diagnosis” may go through multiple diagnostic tasks in the form of a flowchart, and the selection of the next task to be performed may depend on the outcome of the previously accomplished task. The Task Library is a repository of specialized system tasks that provides the instructions to “self-diagnosis” for performing the diagnostic tasks.

To get the details of the problem, “problem detection” sends a diagnostic request to the “self-diagnosis” module that communicates with the Task Library to retrieve the diagnostic tasks to be used to discover the causes of the problem. For the example of the code page problem, the diagnostic request can be “run a code page’s diagnosis” and the diagnostic tasks can be “read the character code in the document and lookup the character code table to find the code page”. The result of the diagnosis is returned to the “problem detection” module for formulating a problem statement that includes all the details of the problem. The search in the “self-knowledge” or in the knowledge base will return a solution on how to fix the problem. The solution of the problem is sent to the “problem resolution” component to be applied to the system. The interactions between the self-diagnosis module and other elements of the autonomic system architecture are shown in Figure 4.

## 2.7. Problem resolution

The role of the “problem resolution” module is to apply the solution received from “problem detection” and to initiate a diagnostic process to verify that the applied solution has fixed the problem of the system. When the “problem detection” module finds a solution for a failure or an abnormal behavior of the system, it sends a problem solution record to the “problem resolution” module to initiate the problem resolution cycle. The “problem resolution” module creates an action record and performs the required operations to fix the problem. For the code page example, the action record can be “download the code page from the code page service site and install the code page”. After the solution is applied, the “problem resolution” module creates a diagnostic request and sends it to the “self-diagnostic” module to verify that the applied solution works properly and has fixed the problem. The “problem detection” module returns the result of the verification, which can be a success or a failure, to the “problem detection” module.

If the “problem detection” module cannot find the problem resolution, a new problem report is generated and stored in the knowledge base. The human expert will be notified and alerted to the new problem report, will resolve the new problem, and will create a problem report and solution record. The problem report will contain a problem classification based on a taxonomy perceivable by programs, a problem description searchable by the

search engine, detailed data required for diagnosis, a recommended sequence of actions for the solution, etc. This problem report is posted to the technical support knowledge base that will notify the affected systems waiting for the solution to resolve the pending problem. The “self-monitoring” module will receive this notification that will trigger the process cycle of the “problem detection” module to fix the problem.

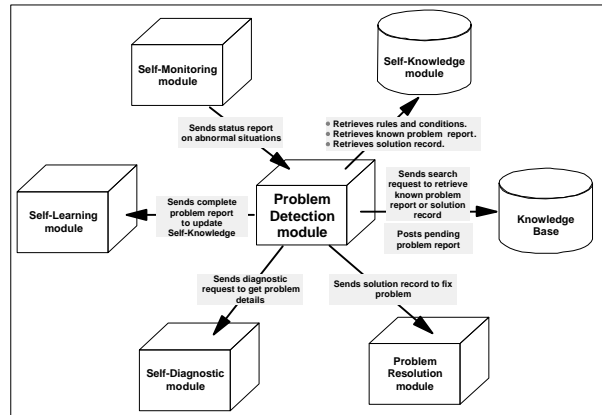


Figure 3. Problem detection module: interactions

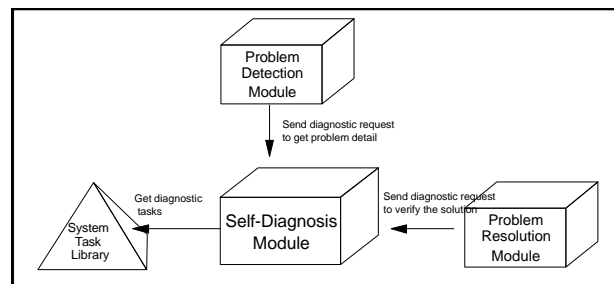


Figure 4. Self-Diagnosis Module: interactions

## 3. Research direction

The challenges of autonomic computing to produce self-healing and managing computers will also bring forward a lot of exciting opportunities for researchers. Finding a solution to a technical problem within the ocean of existing technical information and knowledge is already challenging enough. Today, technical support information search and delivery research tries to extract the most relevant solution to customers’ problems. In accomplishing that, it is always assumed that either the customers or the technical support people are the end users. The content authoring, data storage, information retrieval and delivery systems are designed based on this assumption.

If the consumer of technical support information is not a human being, then technical support information search and delivery systems will undergo a dramatic change. The technical solutions will be delivered to machines and they

will be acted upon without intervention. This will require developing new methods of content generation. The designer of the systems will have to identify the points and functions to be monitored as part of their design. Applicability of the predictive data analysis techniques needs to be tested in predicting impending system failures.

One of the most challenging aspects of autonomic computing will be to determine with confidence one single solution that will fix the problem rather than expecting a human intervention to decide which solution is most relevant. Adaptive learning techniques will be used intensively to make a system self-learning. In contrast to what we experience today, the machines and the systems of the future will get better as they are used.

#### 4. Acknowledgements

We are grateful to Ricardo Vilalta for his ideas and support.

#### 5. References

- [1] G. Petrash. Managing Knowledge Assets for Value. *Knowledge-Based Leadership Conference*. Linkage, Inc. Boston, October, 1996.
- [2] J. Gray. Why do computers stop and what can be done about it. *Symposium on Reliability in Distributed Software and Database Systems*, 3-12, 1986.
- [3] J. Gray. What next? A dozen remaining IT problems. *Turing Award Lecture*, 1999.
- [4] D. Patterson et al. Recovery Oriented Computing (ROC) : Motivation, Definition, Techniques, and Case Studies. Computer Science Technical Report, UCB//CSD-02-1175, U.C.Berkeley, March 15, 2002.
- [5] J. Hennessy. The future of Systems Research. *Computer*, August 1999 32:8, 27-33.
- [6] Autonomic Computing: IBM's Perspective on the State of the Art Information Technology. <http://www.research.ibm.com/autonomic/manifesto/>.
- [7] IBM eServer iSeries Universal Connection for Electronic Support and Service. SG24-6224-00, *IBM Redbook*, published August-27-2001.
- [8] G. Cugola, E. Di Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *Proceedings of the 20<sup>th</sup> International Conference on Software Engineering (ICSE'98)*, Pages 261-270, Kyota, Japan, Apr. 1998.
- [9] D.S. Rosenblum and A.L.Wolf. A design framework for Internet-scale event observation and notification. In *Proceedings of the Sixth European SoftwareEngineering Conference*, Zurich, Switzerland, Sept. 1997, Springer-Verlag.
- [10] R. M. Balzer. EXDAMS – Extendable Debugging and Monitoring System. In *Proceedings of the AFIPS Conference*, Vol. 34, pages 567-580. SJCC, 1969.
- [11] J. L. Hennessy and D. A. Patterson. Computer Architecture: A Quantitative Approach. *Morgan Kaufmann*, Los Altos, CA, 1990.
- [12] M. L. Model. Monitoring system behavior in a complex computational environment. Technical Report CSL-79-1, *Xerox PARC*, 1979.
- [13] R. Snodgrass. A relational approach to monitoring complex systems. *ACM Transactions on Computer Systems*, 6(2):157-196, May 1988.
- [14] A. Kishon, P. Hudak, and C. Consel. Monitoring semantics: A formal framework for specifying, implementing, and reasoning about execution monitors. In *Proceedings of SIGPLAN'91 Conference on Programming Language Design and Implementation*, pages 338-352. ACM, 1991.
- [15] J. E. Lumpp Jr., T. L. Casavant, H. J. Siegel, and D. C. Marinescu. Specification and identification of events for debugging and performance monitoring of distributed multiprocessor systems. In *Proceedings of the 10<sup>th</sup> International Conference on Distributed Computing Systems*, pages 476-483. IEEE. 1990.
- [16] B. Plattner. Real-time execution monitoring. *IEEE transactions on Software engineering*, 10(6):756-764, November 1984.
- [17] M. H. Reilly. A Performance Monitor for Parallel Programs. *Academic Press*, Boston, 1990.
- [18] J. J. P. Tsai, K.-Y. Fang, and H.-Y. Chen. A noninvasive architecture to monitor real-time distributed systems. *IEEE Computer*, 23(3):11-23, March 1990.
- [19] AIX Version 4.3 Understanding the Diagnostics, *IBM Publications*, SC23-2456.