An Approach to Monitor Application States for Self-Managing (Autonomic) System

Hoi Chan, Trieu C. Chieu
IBM T.J Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
914-784-7741
hychan,@us.ibm.com, tchieu@us.ibm.com

ABSTRACT
Autonomic Computing has gained widespread attention over the last few years for its vision of developing applications with autonomic or self-managing behaviors[1]. One of the most important aspects of building autonomic systems is the ability to monitor applications and generate corrective actions should exceptions occur. The problem lies in those applications where source code is not available and therefore it is virtually impossible to modify the application code to include monitoring functions, or the application code is too tangled with other components which make modification difficult. This hinders the inclusion of autonomic features in many of the legacy applications. In this report, we will describe an approach to build generic monitoring systems for legacy applications.

Categories and Subject Descriptors
D.2.3 [**Software Engineering**]: Coding Tools and Techniques – *object oriented programming*

General Terms
Management, Design, Measurement

Keywords
Autonomic, Self-Management, Aspect , Monitor

1. INTRODUCTION
Developing tools and creating new software engineering methodologies to incorporate self-managing features into applications is an important engineering and research topic. In general, monitoring the states of an application and take appropriate corrective actions should exceptions occur is the most common method of producing self managing behavior. The process of monitoring an application is very application specific and in general requires the monitoring features to be incorporated into the application itself. For many existing applications, application source code is not available and appropriate monitoring features may not be provided.

The use of Aspect-Oriented programming technology [2] allows the monitoring functions to be treated as a concern, developed separately, integrated selectively in applications at development and/or at run time.

OOPSLA'03, October 26–30, 2003, Anaheim, California, USA.
ACM 1-58113-751-6/03/0010.

2. ASPECT-ORIENTED PROGRAMMING FOR CREATING MONITORING SYSTEMS
To achieve self-managing behavior, an Autonomic Manager is created to monitor the application [3]. An Autonomic Manager relies on collecting and analyzing information from the monitored applications, and takes appropriate actions. Specifically, to monitor the state of an application, one needs to monitor the values of its variables as well as the sequence of processes the application has executed. Aspect-oriented software development is an innovative technology for separation of concerns (SOC), or the ability to identify, encapsulate, and manipulate only the parts of software which are relevant to a particular goal, concept or purpose [4] in software development. The techniques of Aspect Oriented System Design (AOSD) make it possible to modularize crosscutting (two concerns crosscut if the methods related to those concerns intersect [5]) aspects of a system. The monitoring function of an application can be viewed as a concern, and developed separately from the main application. This methodology can be used to include monitoring functions in development time, where source code is available. General tool such as AspectJ [6] is well developed and readily available for Java programming language. For existing applications, where source code is not available, the same methodology can be applied, but at object code level. For Java application, before all classes (as class files) are loaded into JVM, these classes can be intercepted, analyzed, decomposed, and appropriate non-invasive (without affecting the logic) constructs are added in the Java byte code to provide state information to an external entity, usually the values of the variables at method entry and exit points. An example of the tool which provides such capability is IBM's HyperJ [7] which supports "multi-dimensional" separation and integration of concerns in standard Java software. In addition, the monitor concern can be subdivided into many sub-concerns, each can be switched on and off, allowing the monitoring manager to monitor the concerns which are relevant.

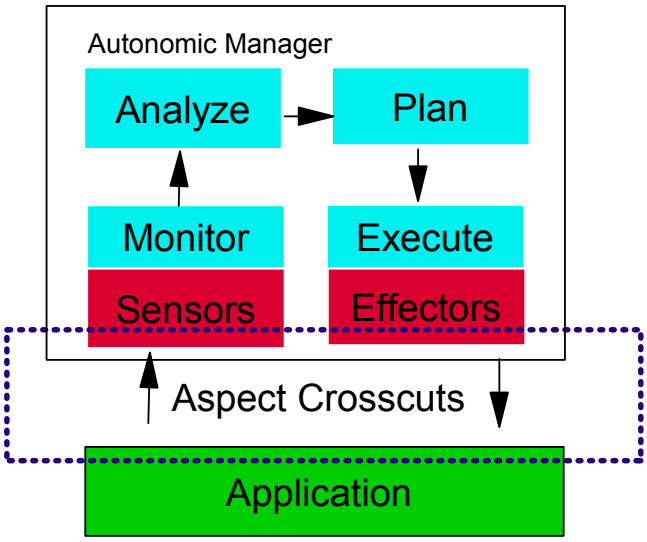

Figure 1: conceptual view

Figure 1 illustrates the building blocks of an Autonomic Manager. An Autonomic Manager [2] consists of the following functions: sensing or information collection, analysis of the collected information, planning actions based on result of analysis, and executing the actions. The application being monitored is connected to the sensors and effectors of the Autonomic Manager via an Aspect crosscuts layer.

3. EXAMPLES

Suppose we want to build an autonomic management system which manages the utilization of a storage device and takes action should utilization reach a predefined value. We want to have Utilization objects monitored by StorageDevice objects, where Utilization is an existing class. Following the Aspect methodology, the implementation is relatively straightforward, an instance *utilizationMonitor* is introduced into the class *StorageDevice,* that keeps track of the *StorageDevice* object which monitors *Utilization*. Monitors are added and removed with the static methods addMonitor and removeMonitor. The pointcut "newData" defines the entities which need to be monitored, and the "handleMonitorValue" defines what we want to do when a change is detected. As we can see, neither StorageDevice nor Utilization's code needs to be modified, and all the additional code required to support this monitoring capability are *within* the Aspect. The following code fragment illustrates the relatively simple steps to generate the required concerns [9].

```
Aspect utilizationMonitor {
  Private Vector utilizationMonitors = new Vector();
  Public static void addMonitor(Utilizaton u,  StorageDevice(s)) {
     utilizatioMonigor.addMonitor(s);
  }
  Public static void removeMonitor(Utilization u,
StorageDevice s) {
     utilizatioMonigor.removeMonitor(s);
  }

  pointcut newData(Utilization u): target(u) &&
            call(void  Utilizatiou.set*(double));
  after(Utilization u): newData(u) {
     Monitor[ ] monitors = u.utilizationMonitors.toArray();
     for(int i=0; i<monitors.length; ++i) {
          handleMonitorValue(u, s);
      }
  }
  static void handleMonitorValue(Utilization u,
            StorageDevice s) {
          s.handleNewUtilization(u);
   }
}
```

Sample code

4. CONCLUSION

The use of Aspect programming methodology [3,4,5] in building Autonomic Manager to monitor applications without modifying source code provides an important path to connect legacy applications with an Autonomic Manager System. However, with the current available Aspect tools, it is still rather limited in the actual usage. Questions such as copy right, security issues with object code interception and modification, may hinder the further usage and development of this programming systems. However, we believe that the beauty of this programming system lies its clear separation of concerns, and the ability to crosscut concerns without modifying source code, which makes it an ideal candidate for providing monitoring services to applications when such capabilities are not built in.

5. REFERENCES

[1] Jeff O. Kephart, David M. Chess, "The Vision of Autonomic Computing", Computer Journal, IEEE Computer Society, January 2003 issue

[2] Aspect Programming Technolgy: http://aosd.net

[3] Autonomic Manager Toolkit (available 3Q, 2003 for free download... : http://www.alphaworks.ibm.com)

[4] Harold Ossher and Peri Tarr, "Using Multidimensional Separation of Concerns to (Re)shape Evolving Software", Communications of the ACM, vol.44, no. 10, pp43-50, October 2001.

[5] Tzilla Elrad, Mehmet Aksits, Gregor Kiczales, Karl Lieberherr, and Harold Ossher, "Discussing Aspects of AOP", Communications of ACM, vol. 44, no.10, pp33-38, October 2001

[6] AspectJ: http://aspectJ.org

[7] HyperJ: http://www.alphaworks.ibm.com

[8] AspectJ programming guide, Xerox Corp.