RC24256 (W0705-025) May 4, 2007
Computer Science

IBM Research Report

Aspect-Oriented Web Services for Distributed Resource Monitoring in Utility Computing

Trieu C. Chieu, Hoi Chan
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Aspect-Oriented Web Services for Distributed Resource Monitoring in Utility Computing

Trieu C. Chieu and Hoi Chan
IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA
{tchieu, hychan}@us.ibm.com

Abstract

Monitoring resource utilization is an essential task in Utility Computing (UC). Typically, a UC manager with adaptors is used to orchestrate and collect the monitoring results. However, large IT systems, such as those in data centers are required to monitor dynamic resources running in distributed platforms, thus demanding a complex UC manager structure. This paper introduces a novel approach that utilizes a platform specific Aspect-Oriented Programming (AOP) tool to dynamically weave a monitoring Web service into a running resource to enable communication with a UC manager exposed with standard monitoring Web service interfaces. This AO-Web service approach provides a simple yet powerful and effective means for the dynamic monitoring of distributed resources running in heterogeneous platforms.

1. Introduction

Utility computing [1] promises to give IT department a means to provide optimized resources in an on-demand basis as business conditions change. This approach enables flexible utilization of IT resources at lower operating costs. However, if moving from a traditional computing model to Utility Computing requires building a whole new infrastructure, there will be few takers due to high implementation and deployment costs. Thus, one of the most important factors in deploying Utility Computing is the ability to incrementally add intelligent management functions to existing IT resources and connect them to a utility manager. However, the key problem lies in those existing resources running in distributed environment where source code may not be available for modification in order to introduce the necessary monitoring functions.

In this paper, we propose to treat the monitoring functions as additional concerns, and use Aspect-Oriented Programming tool [2, 3] to dynamically weave them into the applications at build time and/or dynamically at runtime. To address the issue of connecting to distributed resources, we introduce a standard monitoring web service interface to facilitate interoperability without concerning platform dependency. This approach also offers a novel technique to dynamically integrate and weave new services into existing applications to meet unforeseen post-deployment requirements.

2. Aspect-Oriented Web Services

Aspect-oriented programming (AOP) is a software engineering approach for separation of concerns, or the ability to specify, encapsulate, and manipulate only the parts of software which are relevant to a particular goal, concept or purpose [3, 4]. The techniques of AOP system design make it possible to modularize crosscutting concerns in an application. Two concerns crosscut if the methods related to those concerns intersect [5] different aspects of a system. General tool such as AspectJ [6] is well developed and readily available for Java at development time. For legacy applications where source code is not available, the same methodology can be applied, but at object code level. For Java, the object classes are intercepted, analyzed and decomposed before they are loaded into JVM, and appropriate non-invasive (without affecting the logic) constructs are then inserted by a dynamic aspect weaver to provide appropriate information to an external entity. The provided information usually consists of the values of variables at method entry and exit points. Examples of such Java byte-code tools are AspectWerkz [7] and HyperJ [8].

In a typical Utility Computing (UC) system, a UC manager relies on collected information from monitored resources, and takes appropriate actions based on business rules. Particularly, to monitor the state of a resource, one needs to observe the values of its parameters as well as the sequence of processes that the application has executed. Since the monitoring function of an application can be considered as a different concern, it can be developed as a separate aspect from the main application. Using an AOP tool, the monitoring functions can thus be introduced in development time, where source code is available.

Often, the use of AOP techniques to weave functions into an application is under the assumption that they are deployed on the same platform. For a virtual data center where resources may be distributed across multiple locations and run in different platforms, monitoring these heterogeneous systems requires a different and more flexible approach. We introduce an approach which dedicates the UC manager to expose a standard monitoring Web service interface, and assumes the weaving of the web service as an aspect crosscut in each of the monitored resources. Since Web services use HTTP, WSDL and XML messaging standards to provide connectivity and interoperability between diverse

components, functionality can be provided regardless of how and where the service is implemented.

Figure 1 illustrates a conceptual view of how existing resources in distributed client applications use AOP and web services to interact with the UC Manager for monitoring. Each client resource uses an AOP crosscut defined for its platform to weave a Web service to expose its resource parameters or operating results (e.g. average, max or min usage over a period of time) to the UC Manager. The resource data is typically embedded in the payload of the monitoring Web service request. Upon receiving the request, the UC manager will execute a corresponding service method to process the data.

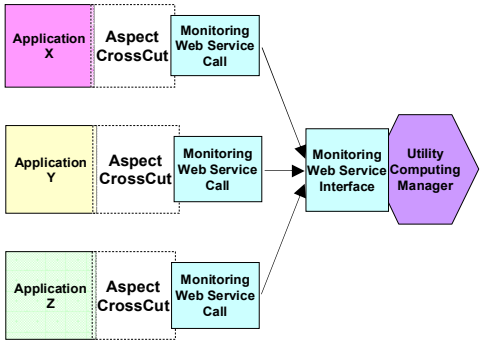

Figure 1. Conceptual view of AOP crosscuts with web services

3. Distributed Resource Monitoring Example

Our example scenario consists of a UC Manager to monitor the utilization of a set of storage devices and takes actions when utilization reaches a predefined value in a virtual data center. Utilization of a device is defined as the number of connections multiplied by a utilization factor. For Java implementation, a "*Device.connect(..)*" method of the "*Device*" object is called inside a client application whenever a new connection is created. There is no existing API for the UC Manager to access this connection information. Following our AOP approach, we define a "*Utilization*" aspect with a global variable "*numberOfConnections*" declared externally for the "*Device*" object to track the number of connections when the "*Device.connect(...)*" method is called. This information is then delivered to the UC Manager via a monitoring Web service call invoked inside a "*newConnection*" pointcut as shown in Figure 2 in AspectJ syntax [6]. Particularly, after returning from the "*Device.connect(..)*" method call, the global variable "*numberOfConnections*" is incremented and a web service call is set up to invoke the "*setUtilizationFactor(..)*" service method of the UC Manager monitoring services. Note that the original "*Device*" object source code is not required, and all the additional code used to support this monitoring capability is declared within the "*Utilization*" aspect. For other remote resources running on different platforms such as Microsoft .Net framework, similar aspects can be implemented using the platform-specific aspect tools to allow cross-platform interoperability.

```
public aspect Utilization {
   public static final long Device.FACTOR=0.2;
   public static long Device.numberOfConnection=0;
   // Define "newConnection" pointcut
   pointcut newConnection(Device d):target(d)&&call(*Device.connect(..));
   // Define after advice for "newConnection"
   after(Device d) returning: newConnection(d) {
     // Increment number of connections and calculate "utilFactor"
     d.numberOfConnection++;
     long utilFactor = d.numberOfConnection*d.FACTOR;
     // Set up Web Service call to UC Manager
     WSIFDynamicPortFactory portFactory = new
WSIFDynamicPortFactory(WSIFUtils.readWSDL(null,wsdlLocation),null, null);
     WSIFPort port = portFactory.getPort("SOAPPort");
     WSIFMessage input = port.createInputMessage();
     input.setPart("utilFactor", new WSIFJavaPart(String.class, utilFactor));
     WSIFMessage output = port.createOutputMessage();
     // Make "setUtilzationFactor" service call
     port.executeRequestResponseOperation("setUtilizationFactor", input,
output, null);
   }
}
```

Figure 2. Sample code fragment for a utilization aspect with web service call

4. Future Work

To facilitate platform-independent development, interoperability and utilization of our aspect-oriented Web service approach on diverse platforms, we are exploring an XML-based AOP tool that can directly manipulate Web services and support dynamic binding of service endpoints via runtime configuration. We will continue to enhance the design by implementing a set of standard and extensible monitoring and management operations in the context of Utility Computing. Also, we plan to apply the approach to a real case of Utility Computing scenario in a data center to validate its effectiveness.

References

[1] "Utility Computing", IBM Systems Journal, vol.43, no.1, 2004.

[2] H. Chan and T.C. Chieu, "An Approach to Monitor Application States for Self-Managing (Autonomic) Systems", OOPSLA'03, Anaheim. California, USA, October 2003.

[3] Aspect Programming Software Development. http://aosd.net.

[4] H. Ossher and P. Tarr, "Using Multidimensional Separation of Concerns to (Re)shape Evolving Software", Communications of the ACM, vol.44, no. 10, October 2001, pp. 43-50.

[5] Elrad et. al., "Discussing Aspects of AOP", Communications of ACM, vol. 44, no.10, October 2001, pp. 33-38.

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

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

[8] AspectWerkz. http://aspectwerkz.codehaus.org.