Development of Generic Probes for Functional and Extra-Functional Diagnosis

Marouane Himdi
KEREVAL Laboratory Test Engineering
Triskell Project – Irisa (University of Rennes I)
Rennes 35700 FRANCE
marouane.himdi@kereval.com

Abstract

In addition to detection of errors related to design, coding or deployment of an application, the diagnosis is a well-known technique for understanding the behavior of a software system and an absolute requirement for its improvement. Unfortunately, applications become more difficult to diagnosis as functionalities provided become complex.

This paper explores the use of dynamic probes that will be injected into running system to collect various information. The innovative aspect of this approach is the use of generic probes to develop diagnosis framework .

1. System Architecture for Diagnosing Applications

The architecture of the diagnosis system will be a distributed one, with a central element (console) which will administrate the different probes [figure 1].

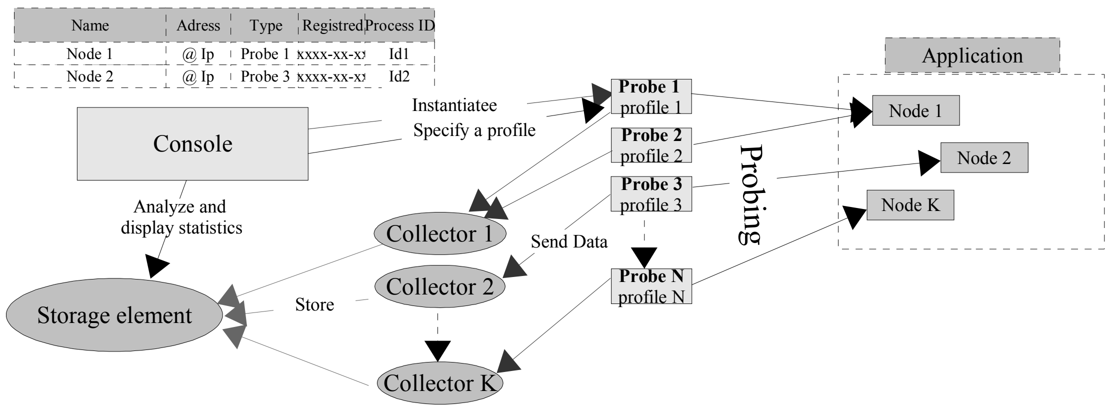
Many categories of probes can be implemented for diagnosing various aspects: network (protocols, etc.), QOS (trafic, etc.), security (intrusion detection, etc.) and system (Cpu, memory, etc.). All these categories of probes will be implemented into each node of the application and the idea is to instantiate the category we need for a given diagnosis.

We will also define a profile which will specify filter criteria used by a probe while collecting data. This will reduce information processed by the storage element. The collector should serve as the first point of registration. Probe will write intercepted messages in a lightweight format to avoid overloading process in which it belongs.

The collector will write messages in a standard format before being processed by the storage element. This one can be a global database viewable in real time to permit the analysis of system/application performance.

2. Component Based System

Component-based system construction has emerged as a fundamentally important technology for software engineering. The difficulty in diagnosing that systems depends on the component model used to implement them. The introspection capability is one of some features required to diagnose components.



Name	Adress	Type	Registred	Process ID
Node 1	@ Ip	Probe 1	xxxx-xx-xx	Id1
Node 2	@ Ip	Probe 3	xxxx-xx-xx	Id2

Figure 1: System diagnosis architecture

3. Injection of Probes

Three levels of diagnosis are offered: a component diagnosis, a middleware diagnosis and finally a system diagnosis [figure 2]

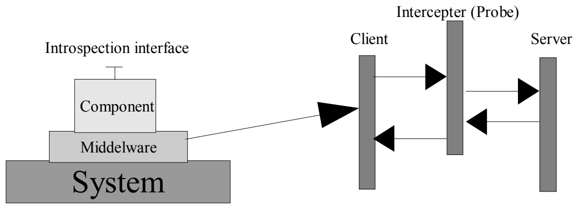


Figure 2: diagnosis levels

1. Component diagnosis:

The component must provide an observable interface (introspection capability), to allow requesting information about execution context and the probe can be directly connected to this interface. In the Fractal component model [1], introspection capability is provided by *Component* and *ContentController* interfaces.

2. Middleware diagnosis:

At this level, we will use the opportunities provided by the middleware (JVM, CLR, ORB...etc) to perform diagnosis. For example, in a Corba environment, probe can be built by using interceptors provided by most ORB [2].

3. System diagnosis

The goal here is to access the memory of the process application and to intercept all calls transmitted by this application. On Windows platform, this can be done by using one of these three techniques [3]: the first called "System-wide hook" is based on the use of windows hooks, but unfortunately it works only with processes linked to USER32.DLL. The second and third techniques are based on the *CreateRemoteThread* method [4] which creates a thread in a target process and then code can be injected by using *loadlibrary* or *WriteMemoryProcess* functions [4]. On the Unix platform, an interesting way is based on the use of some APIs [5] which allow dynamic instrumentation to trace end-to-end call chains.

At this level we will also be interested by the process execution context like Cpu and memory used. This kind of information can be recovered by reading files from the "proc" repository on a Unix plateform .

4. Diagnosis procedure

Relevant information depend on the kind of diagnosis we process.

For a diagnosis system, we have to deal with data as, Cpu, memory in use, etc. . LEWYS [6] is one of those works which aims to built probes to pump information from the system.

For a network security diagnosis, we can look forward to use non-intrusive probes. We are interested about the project SNORT [7] and we are thinking about the use Snort sensor as a probe.

Our approach is to benefit from the wide spectrum of diagnosis tools, and try to adapt techniques used in them to built our probes.

5. Conclusion and future work

In this paper, we try to describe some guidelines to built a diagnosis system based on generic probes injection. We have been inspired by intrusion detection and network monitoring tools.

Future work will focus on thinking about an automated system for correlating and analyzing all of the data gathered to locate and identify possible failures origin.

This topic is the subject of my thesis which aims to develop a diagnosis tool using components based system.

6. Acknowledgments

I would like to thank Alain Ribault and Yves LeTraon for their guidance and advices.

I'm also grateful to Jim Steel from Triskell Project and Thierry Boullet from Kereval laboratory for reading this paper.

7. References

[1] E.Bruneton, T.Coupaye & J.B.Stefani. "The Fractal Component Model". Section 4 "Introspection and intercession". http://fractal.objectweb.org
[2] Todo Scallan "Monitoring and Diagnotics of Corba Systems". Java Developers Journal.com . P 138 -144 juin 2000.
[3] R.Kuster. "Three ways to inject your code into another process.http://www.codeguru.com/Cpp/W-P/system/processesmodules/article.php/c5767/
[4] Msdn library for Windows. http://msdn.microsoft.com/library
[5] L.DeRose, T.Hoover and J.K.Hollingsworth. "The Dynamic Probe Class Library – An Infrastructure for Developing Instrumentation for Performance Tools".
[6] LEWYS "Lewys is Watching Your System" http://lewys.objectweb.org.
[7] SNORT, http://www.snort.org