Dynamic Collection of Reliability-Related Data and Reliability Evaluation for Internet Software

GUO Yong
School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
Email: guoy@hit.edu.cn

MA Pei Jun, SU Xiao Hong
School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
Email: {ma, sxh}@hit.edu.cn

***Abstract*—Internet software is an emerging software paradigm. Traditional evaluation methods for software reliability are no longer applicable because of the open and dynamic characteristics of the Internet software. In order to evaluate its reliability accurately, there must be a dynamic and open reliability evaluation approach. This paper presents the collection approach of the reliability-related data for the Internet software and its components. The paper also designs a software system for the reliability evaluation. The approach bases on the characteristics of Internet software; it uses aspect-oriented programming and pattern programming techniques to trace component real-time running data and save the data to a database. The approach can trace different granularity data according to the performance requirements of the system. The reliability evaluation system can predict the components and system reliability using the real-time data. This makes the components be selected expediently when the Internet software system is assembled dynamically. A case study is presented to illustrate the effectiveness of this approach.**

***Index Terms*—Internet software, reliability evaluation, data dynamic collection, component, Aspect-Oriented Programming**

I. INTRODUCTION

With the development of the Internet, traditional software structure cannot meet the open and dynamic network environment which has unpredictable behaviors and autonomy nodes. In order to meet the challenge, a new emerging software paradigm, Internet software is developed. Ref. [1] gives the definition of the Internet software: "Essentially, Internet software is constructed by a set of autonomous software entities distributed over the Internet, together with a set of connectors enabling the collaboration among these entities in various fashions. The Internet software entities are able to be aware of the dynamic changes of the running environments, and continuously adapt to these changes by means of structural and behavioral evolutions." From the definition we can see, as a new paradigm, Internet software has many different features compared with traditional software. The specific features are autonomous, cooperative, reactivity and multi-objective evolutionary [2]. These features lead to the entity behaviors of Internet software are unpredictable in the Internet environment. Traditional evaluation methods for software reliability fail to satisfy the requirements of the Internet software. It needs a new dynamic method to evaluate reliability of each unit and Internet software system [3].

The structure of traditional software will not change automatically after it has been developed and put into use[4]. Contrarily, at different times, the Internet software may use different components to finish the same function. The constitution paradigm of the Internet software is no longer a static tightly-coupled, but a dynamic assembly way. It will adjust and evaluate itself automatically according to the requirements of the function and reliability after perceiving changes in the external environment [1]. Sometimes a system needs the collaboration between components' service to implement its required function. To ensure the function implementation, the selection of components not only depends on whether the components provide the desired services, but also depends on the reliability of the services.

Traditional approaches usually only give the overall reliability of a component [5-7], but do not give the reliability of each service. Therefore, it can't be known whether the component meets the dynamic assembly requirements. For example, two components provide the same required services. The overall reliability of the first component is better than the second one. However, the reliability of the first component services used in the designed system is worse than the second one. Therefore, the second component should be selected. If a component provider does not provide the service reliability, a component user possibly makes the wrong choice select the first component. So if a user wants to make a right selection from many meeting requirement components, it is not enough only to know component reliability; the services' reliability of a component should be obtained.

Manuscript received May 22, 2012; revised October 25, 2012; accepted November 18, 2012.
Corresponding author: GUO Yong, Email: guoy@hit.edu.cn

doi:10.4304/jsw.8.6.1390-1397

However few studies focused on how to obtain the reliability of the component services before. The reliability evaluation of component service needs to accumulate a series of historical data from the real running environment, because there is a certain difference between the reliability of the emulation environment and the actual running environment. Moreover, designing the emulation environments for some complex systems are very difficult. Therefore, we should better trace each component service in the actual environment to obtain the information, and then evaluate the reliability. Just like Ref. [8] mentioned, "The reliability evaluation of the Internet software emphasizes a flexible dependability evaluation, inference and application mechanism based on historical information running in an open environment." In order to achieve this goal, this paper proposes a dynamic data collection approach and an evaluation framework for the Internet software reliability.

The paper proposes an approach which uses AOP and pattern programming to select the appropriate connection point according to the evaluation granularity. The approach monitors the use of components and records the corresponding reliability data in real-time. Component users can obtain component reliability from the framework and can predict the running system reliability at a certain time. The trace type includes operation, error and other related information. The reliability evaluation results can be used to select the component in the dynamic assembly. The proposed method makes the collection of reliability data with the authenticity, continuity and integrity.

The rest of the paper is organized as follows: Section II presents works related to studying. Section III introduces the main process. Section IV discusses the dynamic collection approach and reliability evaluation. In Section V, we give a demonstration and performance analysis. Finally, we conclude this study.

II. RELATED WORK

Some researchers proposed techniques for monitoring software executions[9]. Ref. [10] proposed a method to generate tests for single components and for their integration automatically. The method focuses on test and integration; it is unconcerned with reliability data collection. Ref. [11] gave a method that could automatically analysis methods in the bytecode, but it is only used for java language. Ref. [12] presented an infrastructure for monitoring and managing distributed middleware, but the method is a bit complicated for the dynamic collection of the Internet software data. Bertolino proposed approaches for tracing dependability and performance of connected systems [13]. It focused on dynamically connected systems. Ref. [14] proposed an extension for the conventional dynamic data flow analysis to test Java programs. It focuses on Java programs.

Recently, there have been many literatures about how to apply AOP technology to trace the behaviors of software system and test software reliability [15-18]. Ref. [15] focuses on fault detection and recovery using AOP technology. Ref. [15] proposed a method to obtain the reliability information dynamically in a software system using AOP, but the trace granularity is bigger. The method only concerns the reliability of the entire component. It neither concretely concerns the used behaviors' reliability of a component, nor records the interaction information of each component's behavior. Ref. [16] focus on the collection of software maintainability dynamic metrics using AOP. Ref. [17] mainly studied how to use AOP for automatic testing. Ref. [18] mainly concerns the reliability of the design phase. These methods do not address how to collect the running data and evaluate reliability in the open Internet software environment detailed. Ref. [16] presented an AOP-based framework for collecting dynamic metrics, it only suits for the component programed in java language.

III. MAIN PROCESS OF THE PROPOSED METHOD

Generally speaking, a component has a reliability index after it has developed. The index is usually assessed in the test environment. The test value will inevitably have the deviation from the value coming from the real running environment, because the test environment is different from the actual running environment. Therefor we should collect the data from the real running environment, dynamically measure the component reliability based on the collected data. Dynamic reliability assessment is based on the idea; data are collected automatically while the component is used in the real environment. The main process is shown in Fig.1.

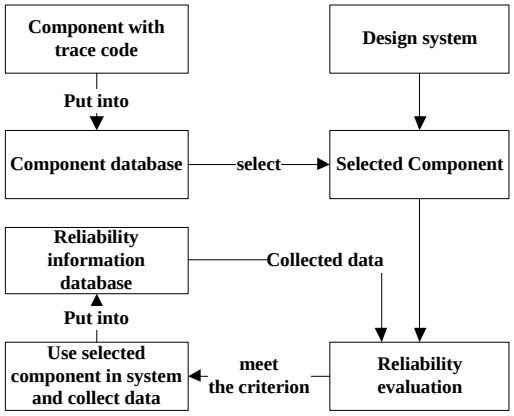


Figure 1. The selection and information collection for component

A component combined with its trace code is put into the component database. Component users select a component form the database when they design system. Component users evaluate the reliability of the selected component. If the reliability does not come up to the requirements, they select another one. When the designed system is running, trace code will collect the running information and store them into the reliability information database automatically. Component users will use these collected data to select component when they design system or used for Internet software dynamically assembling. This paper mainly discusses how to program the trace code and the way of using collected data.

IV. DYNAMIC COLLECTION OF RELIABILITY DATA

A. Granularity of data collection

A component is usually implemented using object-oriented technology. Component and object are abstract descriptions of the real world, which encapsulate reusable code. However, the object provides methods to a user, and the components provide services to a user. A component may contain one or more objects. The component services are ultimately achieved through some concrete methods of the objects inside the component. So the reliability-related data collection can be divided into different granularity according to the concrete circumstances. It can be divided into component-level, object-level, service level and method level. The collected information can include component's ID, component running time, the called object of the component, the services of the component, the methods of the object ,their success or failure information and so on.

The data collection of the component-level regards the component as a whole and its internal structure is not concerned. Only the two processes are traced; the two processes are the reference of the component and the completion of the assigned functions. The collected data include the times of success and failure, the component running time, the interaction between components and so on. The component reliability is evaluated based on the collected data.

The reliability data collection of object-level is more fine-grained. This way traces the use of the objects of a component. The trace includes the interaction between the objects, the times of success and failure, and so on. Generally speaking, the objects in a component are packaged together, so that the communication reliability cannot be considered, namely, communication among objects in a component is reliable.

The data collection of service-level means monitoring the use of each service and collecting the related information. Many components not only provide one service, so obtaining the reliability of each service is more meaningful. Some systems only use some of the services of a component, not all the services, so we should use the actually used services of a component to evaluate the system reliability.

We can trace each method of the object in a component and collect its running information. Using this method, we can obtain the called frequency and locate faults for each method. The traced methods include the external and internal methods. The external method is the "public" method and is used for objects to call one another in a component, and it can be defined as a service of the component. The internal method is "private" or "protected" method in an object. The internal method cannot be called direct by other objects. It can only be called within its object.

The choice of the collection granularity depends on the actual situation because the data are collected in a real running environment of a system; the trace code may affect the performance of the system. We should consider the following aspects: First, efficiency, if the system requires a high operating efficiency, we cannot use more fine-grained data collection methods, otherwise the efficiency will be decreased and affect the system operation. The second is the available information of the component. If the internal structure information of the component is not available, we had better use component-level or service-level collection method. Otherwise we can use object-level or method-level collection method. No matter what kind of granularity is used may affect the performance of the system. In order not to affect the actual use of the system, we should choose the granularity according to the performance requirements of the system.

Bytecode instrumenting tools [11] or middleware functionality [12] can be used to collect running data, but the two methods are a bit complicated. We will use the adapter pattern [19], proxy pattern [19] and weaving method for online data collection. These three methods are more convenient. After the data collection, codes are combined with a component, if only the internal codes of the component are modified, the data collection codes will not require modification. We can simply replace the original component with the new one.

B. Data collection using adapter

The adapter pattern [19] can translate one interface for a class into a compatible interface. By this way, the components which interfaces are incompatible can work together. The adapter pattern is often referred to as the wrapper pattern or simply a wrapper. Data collection is achieved by a wrapper of a component. We use a wrapper wrap the component so that the component provides its services to outside through the wrapper. We put the tracing code in the wrapper avoid modifying the component code. This way can be used COTS components and other components. The component wrapping diagram is shown in Fig. 2. This method can collect service-level and component-level reliability information.

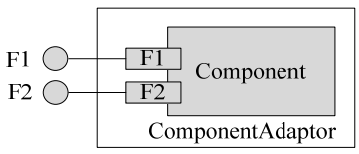


Figure 2. The wrapped component

A typical code structure is as follows.

```
public class ComponentAdapter implements Target
{
   private Component adaptee;
   Private ReliabilityTrace reliabilityTrace;
   public ComponetnAdapter (Component adaptee)
   {
       this.adaptee=adaptee;
   }
   public void F1()
   {
    reliabilityTrace.before();
    adaptee.F1();
```

```
    reliabilityTrace.after();
  }
  public void F2()
  {
    reliabilityTrace.before();
    adaptee.F2();
    reliabilityTrace.after();
  }
}
```

The "ReliabilityTrace" is a class. This class is used for trace component running information of different granularity and storing them into a database. The "Component" is wrapped by "ComponentAdapter."

C. Data collection using proxy

Proxy pattern [19] provides a proxy object for a single object, and the proxy object controls the reference of the object. We can put tracing code in the proxy object and avoid modifying the component code. However, we have to design a proxy for every component and put some necessary code. The workload is bigger. Component proxy diagram is shown as Fig.3.

A typical code structure is as follows.

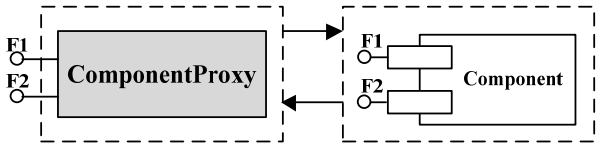


Figure 3. **Component proxy diagram**

```
public interface ComInterface
{
  public void F1();
  public void F2();
}
public class ProxyComponent implements ComInterface
{
  Component realCom=null;
  public void F1()
  {
    if (realCom == null)
    {
      realCom = new Component ();
    }
        reliabilityTrace.before();
    realCom.F1();
    reliabilityTrace.after();
  }
  …
}
```

D. Data collection using AOP

Aspect-Oriented Programming(AOP)[20] is an extension of the object-oriented paradigm. AOP uses "crosscutting" technology to encapsulate the common behaviors into the called "aspect" reusable modules. The common behaviors usually are not relevant to the business and impact many classes. AOP reduces code duplication and coupling between modules in a system. It is beneficial to system maintainability and maneuverability. According to "cross-cutting" technology, AOP divides a software system into two parts: the core concerns and crosscutting concerns. The functions which the system provides are the core concern; another part which has little relationship with the functions is a cross-cutting concern, such as: user authentication, logging, security, etc. There are two ways of the AOP implementation: dynamic crosscutting and static crosscutting. Dynamic crosscutting is implemented by intercepting the object receiving message and replaces the original object's behaviors with the new behaviors. The second is a static crosscutting. Compiler weaves the aspect codes into the original codes when the program is compiled. It does not dynamically change an object's behavior.

Currently, there are hundreds of AOP-related projects. The mainstream program languages, such as Java, C++, C#, etc., support AOP. The java-based AOP tools, which have been adapted for commercial use, mainly include Aspectj, AspectWerkz, SpringFramework and Jboss and so on. AOP includes the following features:

- join point: an execution point
- point cut: a structure used to capture join point
- advice: the execution code of point cut. It is the implementation of "aspects"
- aspect: the composition of point cuts and advice
- introduce: it is used to introduce additional methods or properties for object, which can modify the object structure.

This section will present how to use AOP techniques to get the actual operation information of the components in the open dynamic Internet software environment.

The Aspectj method weaves aspect code into a component. The advantage of this method is that we can obtain more fine-grained information of a component, such as the method running information within a component or an object. The weaving method is shown as Fig.4.

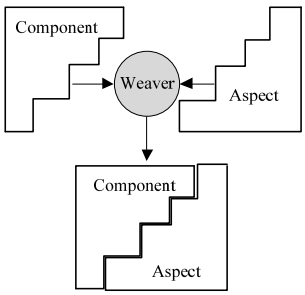


Figure 4. The diagram for weaving method

A typical code structure is as follows.

```
public aspect TraceAspect
{
  pointcut TracePointcut():call(* F1(..))||call(* F2(..);
  before():TracePointcut ()
  {
      reliabilityTrace.before();
  }
```

```
    after():TracePointcut
    {
        reliabilityTrace.after();
    }
}
```

E. Reliability evaluation

A component user can choose an existing model or design their own model for evaluating components' reliability in the system design stage using the collected historical data of the components. Based on the evaluating results, the component user decides whether to use the components. In order to facilitate developer to select components, we can add a variety of typical models such as J-M model [21], NHPP model [22], Musa Basic model [23] to the component management system. We use the NHPP [22] model as an example to describe the calculation method.

NHPP models are widely used for evaluating software reliability. It is an "exponential model". The execution of the Internet software can be modeled as a fault counting process. If the last failure occurred at time t, the software reliability in the time interval $(t, t + x)$ is as follows.

$$R(x \mid t) = e^{-[m(t+x)-m(t)]} \tag{1}$$

$m(t)$ is the expected number of faults which experiences up to a certain time t. $m(t)$ can be calculated using the collected data.

The components in the Internet software distribute on Internet nodes. They are assembled by network connection. The communication reliability impacts the overall system, so the communication reliability must be considered. Suppose $m_s(t)$ is the system expected number of fault to time t then $m_s(t)$ as follows:

$$m_s(t) = \sum_{i=1}^{N_s}[m_i(\tau_i t_c + T_i) - m_i(T_i)] + \sum_{i=1}^{N_s}\sum_{j=1}^{N_s}[d_{ij}(\pi_{ij} t_l + T_{ij}) - d_{ij}(T_{ij})] \tag{2}$$

Where $t = t_c + t_l$, t_c is sum of all components' execution time to the time t, t_l is sum of all components' communication time to the time t. τ_i is the execution time proportion of component i in the time t_c. π_{ij} is the proportion which component i communicates with component j in the time t_l. T_i is the time that component i has executed. T_{ij} is the time that component i has communicated with component j. N_s is the sum of system components.

If the communication is reliable, and we do not consider the internal structure of components, then (2) becomes (3).

$$m_s(t) = \sum_{i=1}^{N_s}[m_i(\tau_i t_c + T_i) - m_i(T_i)] \tag{3}$$

V. Demonstration and Performance Analysis

A. The ATM system for demonstration

OSGi(open services gateway initiative) [24] provides a service-oriented, component-based development approach. Nowadays, some literatures present the development of the Internet software based on OSGi [25, 26]. We will use an ATM (automated teller machine) system to illustrate the method presented in this paper. The system is developed based on R-OSGi platform. The system running information will be captured by the trace code and stored to a database. We develop a component management system to evaluate the reliability of each component and system using the collected information. Fig.5 is the overview of the component register and use.

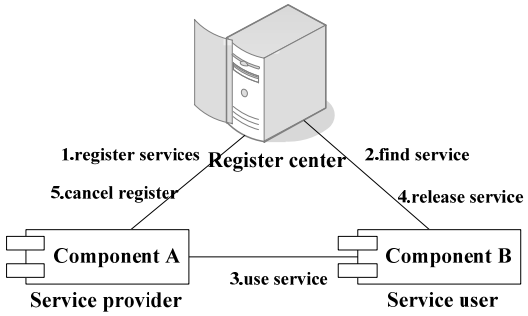


Figure 5. The overview of components register and use

The ATM system includes four components: system main component (ATMmain), security management component (SecurityManager), transaction processing component (TransactionProcessor) and data access component(DataAccessor). The interaction among these components is shown as Fig.6.

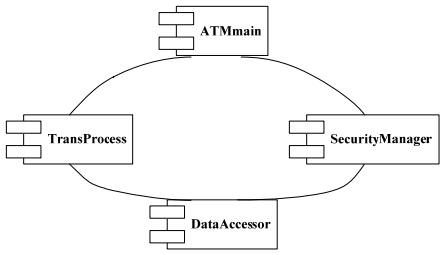


Figure 6. The components relationships in the Atm system

We use the methods presented in this paper to collect the execution information of the system. And then use the collected data to evaluate the system and component reliabilities. Table I shows the services provided by the components.

TABLE I.
SERVICES PROVIDED BY THESE COMPONENTS

ATMmain	SecurityManager	TransProcess	DataAccessor
mainFram	exitCard	Deposit	Update
	encrypt	withdraw	getData
	modifyPwd	Transfer	
	verifyUser	queryBanalce	
	getCardID	printBill	
	decrypt		
	checkPwd		

B. Using Aspectj method for data collection

Using Aspectj method collects the system running information. The class diagram of trace code is shown in Fig.7. The "TraceAspect" is the aspect code for tracing the operational information of each unit, the "Process" is used to process the obtained information obtained, and the "DataAccess" is used to store the processed information to a database. In order to minimize the impact to the system, the collected data are not real-time stored to the database. The collected data are kept in a queue. These data are stored to the database after an ATM user finishes all of his operations.

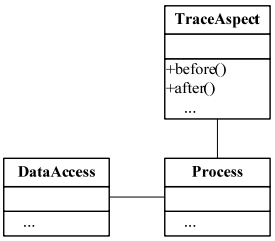
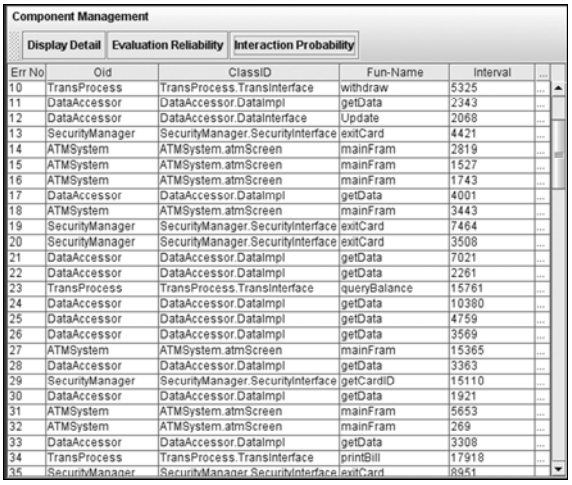


Figure 7. Data collection model using Aspectj

C. Reliability evaluation system

Besides the ATM system, we also developed a system to present how to use the collected data. The developed system can show all collected data, evaluate a component and a system reliability using the collected data, calculate the components' interaction frequencies, use frequency of each component, and performance analysis.

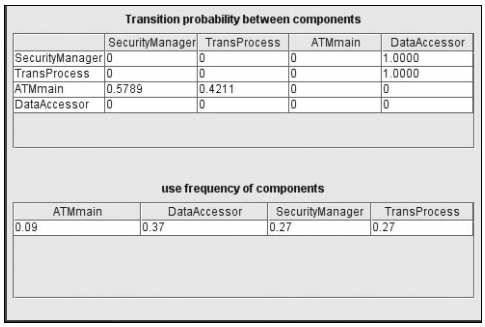
After a period of execution of the system, we accumulate some trace data. These data can be used to evaluate the reliability. Fig.8 shows the failure information of components obtained by trace method proposed above.



Err No	Oid	ClassID	Fun-Name	Interval
10	TransProcess	TransProcess.TransInterface	withdraw	5325
11	DataAccessor	DataAccessor.DataImpl	getData	2343
12	DataAccessor	DataAccessor.DataInterface	Update	2068
13	SecurityManager	SecurityManager.SecurityInterface	exitCard	4421
14	ATMSystem	ATMSystem.atmScreen	mainFram	2819
15	ATMSystem	ATMSystem.atmScreen	mainFram	1527
16	ATMSystem	ATMSystem.atmScreen	mainFram	1743
17	DataAccessor	DataAccessor.DataImpl	getData	4001
18	ATMSystem	ATMSystem.atmScreen	mainFram	3443
19	SecurityManager	SecurityManager.SecurityInterface	exitCard	7464
20	SecurityManager	SecurityManager.SecurityInterface	exitCard	3508
21	DataAccessor	DataAccessor.DataImpl	getData	7021
22	DataAccessor	DataAccessor.DataImpl	getData	2261
23	TransProcess	TransProcess.TransInterface	queryBalance	15761
24	DataAccessor	DataAccessor.DataImpl	getData	10380
25	DataAccessor	DataAccessor.DataImpl	getData	4759
26	DataAccessor	DataAccessor.DataImpl	getData	3569
27	ATMSystem	ATMSystem.atmScreen	mainFram	15365
28	DataAccessor	DataAccessor.DataImpl	getData	3363
29	SecurityManager	SecurityManager.SecurityInterface	getCardID	15110
30	DataAccessor	DataAccessor.DataImpl	getData	1921
31	ATMSystem	ATMSystem.atmScreen	mainFram	5653
32	ATMSystem	ATMSystem.atmScreen	mainFram	269
33	DataAccessor	DataAccessor.DataImpl	getData	3308
34	TransProcess	TransProcess.TransInterface	printBill	17918
35	SecurityManager	SecurityManager.SecurityInterface	exitCard	8951

Figure 8. Failure information of the components

The transition probabilities between components and used probabilities are shown in Fig. 9. These values can be used to evaluate the components and the system reliability when we develop a new system.



Transition probability between components

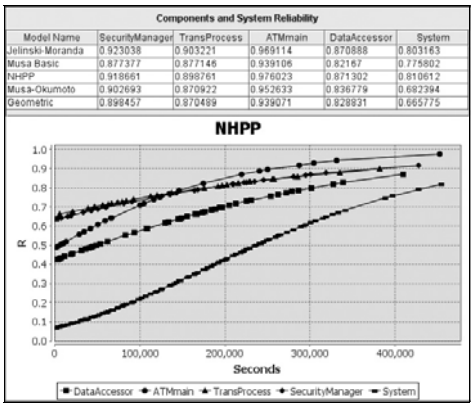
	SecurityManager	TransProcess	ATMmain	DataAccessor
SecurityManager	0	0	0	1.0000
TransProcess	0	0	0	1.0000
ATMmain	0.5789	0.4211	0	0
DataAccessor	0	0	0	0

use frequency of components

ATMmain	DataAccessor	SecurityManager	TransProcess
0.09	0.37	0.27	0.27

Figure 9. The transition probabilities and use frequencies the components

Fig.10 shows the reliability evaluating results of the components and system using some models according to the collected data.



Model Name	SecurityManager	TransProcess	ATMmain	DataAccessor	System
Jelinski-Moranda	0.923038	0.903221	0.969114	0.870888	0.803163
Musa Basic	0.877377	0.877146	0.939106	0.82167	0.775802
NHPP	0.918661	0.898761	0.976023	0.871302	0.810612
Musa-Okumoto	0.902693	0.870922	0.952633	0.836779	0.682394
Geometric	0.898457	0.870489	0.939071	0.828831	0.665775

Figure 10. The evaluating results of components and system reliability

D. Comparison of performance

In order to compare the performance of a system for different trace method with the system which does not have the trace codes, we developed the ATM systems using different trace methods mentioned above. One is no reliability data collection. The other three ones use different methods for reliability data collection. The first one uses Aspectj for reliability data collection, the second uses adapter for reliability data collection, and the third uses proxy method for reliability data collection. We run the four systems in the same environment and compare the performance. The results are as follows.

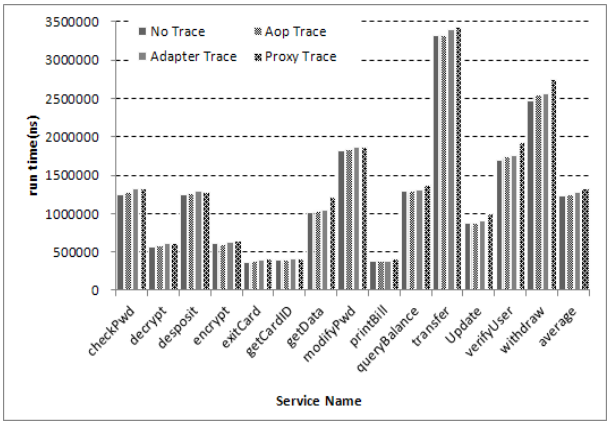


Figure 11. Performance comparison

TABLE II
COMPONENT SERVICES RUNNING TIME IN THE FOUR SYSTEMS

Service Name	No Trace(ns)	Aspectj Trace(ns)	Adapter Trace (ns)	Proxy Trace (ns)
checkPwd	1247423	1274343	1316977	1324885
decrypt	570327	579192	614995	620250
desposit	1251775	1261431	1284827	1285886
encrypt	604690	605586	623545	637982
exitCard	360945	374396	391070	410138
getCardID	399328	402743	404173	412012
getData	1015761	1037171	1038519	1213220
modifyPwd	1822927	1839522	1860790	1869731
printBill	384481	385068	386128	413250
queryBalance	1289444	1300187	1303103	1375828
transfer	3318670	3325806	3401584	3429219
Update	868911	873787	900272	995127
verifyUser	1689182	1743935	1763369	1932543
withdraw	2468135	2552478	2558709	2746145
average	1235143	1253975	1274862	1333301
Impact Rate (%)	0	1.52	3.22	7.95

From Table II and Fig.11 we can see that the Aspectj trace method has the minimal impact on system performance. The average running time of the services in the system which uses the Aspectj trace is only lower 1.52% than the method that has no data collection; the second is the adapter trace method, the average running time of the services is 3.22% slow; the last is the proxy trace method, the service average running time is 7.95% slow. Aspectj trace has the least impact on system performance, but the method requires that the programming language of the components must support the Aspectj technology. The other two methods have larger application scope; especially the adapter trace method is suitable for components written in any program language. However, the adapter trace mode and proxy trace mode will make the data trace code tangle with the component together.

VI. CONCLUSION

Aiming at the Internet software, this paper proposes the automatic dynamic data collection methods and the component evaluation framework. Usually the reliability of commercial-off-the-shelf components is a static value. The value cannot represent the real reliability of the component that runs in the Internet. The components of the Internet software will evolve continuously; its reliability will continue to change. We should collect the reliability-related data continuously, and then we evaluate and predict the reliability of component and system according to the data. In this way we can get more accurate component and system reliability. The collected data would not be accurate enough if the collection is done entirely by hand. Especially the failure time and the execution duration of every component and system would have the deviation with the real value. The method proposed in this paper can collect various information of a system expediently, and can easily evaluate reliability of component and system using collected data. Based on the system performance requirements, this paper gives the data collection methods of different granularity. We can select a proper method according to the actual system performance requirements.

The method presented in this paper is only a preliminary model. The selection of granularity depends on the system information we can get. If we want to use the method-level collections, we must have the source code or component internal information. How to collect method-level information in the absence of source code conveniently needs further study.

ACKNOWLEDGMENT

This research was supported by the National Natural Science Foundation of China (Grant No. 61073052 and No. 61173021).

REFERENCES

[1] H. Mei and X. Z. Liu, "Internetware: An Emerging Software Paradigm for Internet Computing," *Journal of Computer Science and Technology,* vol. 26, pp. 588-599, 2011.

[2] L. Jian, T. Xian-Ping, M. Xiao-xing, H. Hao, X. Feng, and C. Cun, "On agent-based software model for internetware," *Science in China,* vol. (Series E), pp. 1233-1253, 2005.

[3] A. Bertolino, A. Calabró, F. Lonetti, A. Di Marco, and A. Sabetta, "Towards a model-driven infrastructure for runtime monitoring," *Software Engineering for Resilient Systems,* pp. 130-144, 2011.

[4] Z. Li and J. Tian, "An Approach of Trustworthiness Evaluation of Software Behavior Based on Multidimensional Fuzzy Attributes," *Journal of Computers,* vol. 7, pp. 2572-2577, 2012.

[5] S. S. Gokhale, "Architecture-based software reliability analysis: Overview and limitations," *Ieee Transactions on Dependable and Secure Computing,* vol. 4, pp. 32-40, Jan-Mar 2007.

[6] M. Palviainen, A. Evesti, and E. Ovaska, "The reliability estimation, prediction and measuring of component-based software," *Journal of Systems and Software,* vol. 84, pp. 1054-1070, 2011.

[7] K. Li, J. Kou, and L. Gong, "Predicting software quality by optimized BP network based on PSO," *Journal of Computers,* vol. 6, pp. 122-129, 2011.

[8] L. Jian, M. X. Xing, T. X. Ping, X. Feng, and H. Hao, "Research and progress on Internetware," *Science in China,* vol. (Series E), pp. 1037-1080, 2006.

[9] Z. Li and J. Tian, "A software behavior automaton model based on system call and context," *Journal of Computers,* vol. 6, pp. 889-896, 2011.

[10] L. Mariani and M. Pezze, "Behavior capture and test for controlling the quality of component-based integrated systems," 2003, pp. 23-28.

[11] H. B. Lee and B. G. Zorn, "BIT: a tool for instrumenting java bytecodes," presented at the Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems, Monterey, California, 1997.

[12] G. Rackl, M. Lindermeier, M. Rudorfer, and B. Süss, "MIMO — An infrastructure for monitoring and managing distributed middleware environments," presented at the IFIP/ACM International Conference on Distributed systems platforms, New York, New York, United States, 2000.

[13] A. Bertolino, A. Calabró, F. Di Giandomenico, and N. Nostro, "Dependability and Performance Assessment of Dynamic Connected Systems," *Formal Methods for Eternal Networked Software Systems,* vol. 6659, pp. 350-392, 2011.

[14] A. Boujarwah, K. Saleh, and J. Al-Dallal, "Testing Java programs using dynamic data flow analysis," 2000, pp. 725-727.

[15] P. Arpaia, M. L. Bernardi, G. Di Lucca, V. Inglese, and G. Spiezia, "An Aspect-Oriented Programming-based approach to software development for fault detection in measurement systems," *Computer Standards & Interfaces,* vol. 32, pp. 141-152, 2010.

[16] A. Tahir and R. Ahmad, "An AOP-Based Approach for Collecting Software Maintainability Dynamic Metrics," in *Computer Research and Development, 2010 Second International Conference on*, 2010, pp. 168-172.

[17] A. Dantas, F. Brasileiro, and W. Cirne, "Improving automated testing of multi-threaded software," *2008 First IEEE International Conference on Software Testing, Verification and Validation (ICST '08),* pp. 521-4, 2008.

[18] F. Rafique, K. Mahmood, U. R. Tauseef, and K. Rasheed, "Design Phase Analysis of Software Reliability Using Aspect-Oriented Programming," in *Information and Communication Technologies, 2005. ICICT 2005. First International Conference on*, 2005, pp. 263-271.

[19] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," *Reading: Addison-Wesley,* 1995.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier*, et al.*, "Aspect-oriented programming," *ECOOP'97Object-Oriented Programming,* pp. 220-242, 1997.

[21] Z. a. M. Jelinski, P, *Software Reliability Research*. New York: Statistical Computer Performance Evaluation, Academic Press, 1972.

[22] A. L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability,* vol. 28, pp. 206-211, 1979.

[23] J. D. Musa, "A theory of software reliability and its application," *IEEE Trans. Software Engineering,* vol. SE-1(3), pp. 312-327, 1975.

[24] O. Alliance, *Osgi service platform, release 3*: IOS Press, Inc., 2003.

[25] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-OSGi: Distributed applications through software modularization," 2007, pp. 1-20.

[26] Z. Shi and H. L. Peng, "Dynamic Service Evolving Based On OSGi," *Journal of Software,* pp. 1201-1211, 2008.

Yong Guo Since 2008 he has been a PhD candidate in the Department of Computer Science & Technology, Harbin Institute Technology, China. His main research interests include software quality assessment, component-based software development, aspect-oriented programming.

Pei-Jun Ma was born in 1963. PhD, professor and PhD supervisor the Department of Computer Science & Technology, Harbin Institute Technology, China. Member of China Computer Federation. His Main research interests include software engineering, information fusion, image processing, and space computing.

Xiao-Hong Su was born in 1966. PhD, professor and PhD supervisor the Department of Computer Science & Technology, Harbin Institute Technology, China. Member of China Computer Federation. Her main research interests include software defect detection, image processing and reorganization, information fusion, intelligent computing, etc.