A Language to build a Symptom Catalog for Autonomic Computing System

M. M. Math
KLS, Gogte Institute of technology, Udyambag, Belgaum Karnataka

Anjana. N
KLS, Gogte Institute Of Technology,Udymbag, Belgaum

U. P. Kulkarni
SDM College of Engg Technogy, Dharwad

ABSTRACT
The aim of the Autonomic Computing is to increase reliability and performance by enabling the system to be Self- managed. Challenging issue in making the system autonomous is to introduce the standardization for co-existences of heterogeneous system. Every system has its own log file format, to make multiple systems to co exists in the corporate world there is a necessity to follow standard format called Common Base Event (CBE) to represent the log files, the context called Symptom and the actions that administrator may initiate. This paper proposes a new user friendly language to represent the Symptoms and associate the actions and supports the co-existences of systems with heterogeneous log format without the need for conversion to CBE format.

General Terms
Log Trace analyzer, Log Adapter, Autonomic System Manager

Keywords
Autonomic Computing, Self managed resource, log file, Common base event and Symptom Catalog.

1. INTRODUCTION
The advances in computing and communication technologies and software have resulted in an explosive growth in computing systems and applications that impact all aspects of our life [1][2][3]. However, as the scale and complexity of these systems and applications grow, their development, configuration and management challenges are beginning to break current paradigms, capabilities of existing tools and methodologies. Also, these computing systems and applications are unmanageable and insecure. This is beginning to outpace our ability to manage them. This has led researchers to consider alternative approaches based on strategies used by biological systems to successfully deal with similar challenges of complexity, dynamism, heterogeneity and uncertainty.

A general problem of modern distributed computing systems is very complex and managing of such system is becoming a significant limiting factor in their further development. Many companies and institutions are employing large-scale computer networks for communication and computation. The distributed applications running on these computer networks are varied and deal with many different tasks, ranging from internal control processes to presenting web content and to customer support. Additionally, Mobile computing [4] is pervading these networks at an increasing speed. To access their companies' data the employees need to communicate with their companies while they are not in their office using laptops, PDAs, or mobile phones with various forms of wireless technologies.

This develops huge complexity in the overall computer network and administrator finds difficulty in controlling these systems manually. This manual control is time-consuming, expensive, and error-prone and effort needed to control a complex networked computer system tends to increase very quickly towards infrastructure at the client specific application and database layer. Most Autonomic service providers guarantee only up to the basic plumbing layer i.e. up to power, hardware, operating system, network basic database parameters etc.

A possible solution could be to enable modern, networked computing systems to manage themselves without direct human intervention. In an Autonomic System, the human operator doses not control the system directly. Instead, he defines general policies and rules that serve as an input for the self-management process. This has led to the IBM Autonomic Computing initiative.

2. RELATED WORK
IBM's initiation of Autonomic Computing and continuous work over the years has produced 'Autonomic Computing Toolkit' '[5][6][7] which assists software developers in enabling software components to become autonomic. The toolkit also provides the base for developers of management software to be able to manage entities that have been properly enabled. It has tools like Log Trace Analyzer, Generic Log Adapter, Symptom Catalog, and Correlation Engine etc, which can together implement Autonomic solutions.

Symptom Catalog is a repository of the rules which are used for monitoring the application. It maintains the rules and associated steps to be taken when that rule is breached. The present complex system needs a huge knowledge database to operate tools which keeps system administrator away from implementing Autonomic solutions [8]. Also there is a need for easier way of maintaining symptom rules for an administrator is evident.

Autonomic System Manager (ASM) is focused on creating an environment for the system administrator where he can monitor and maintain applications by creating policies for the working of the application. Any breach of policy is undesired, the ASM takes care and alerts the administrator to take appropriate action associated with the rule. The ASM [9] provides the administrator with a user friendly "SYMPTOM CATALOG" (SC) which allows him to create policies for the application. Administrator is endowed with the ability to create simple rules by having rule expressions as combinations of various events and operations between them. The simpler rules can be reused for the creation of complex rules. This is a new feature of SC to provide complex rule creation is what makes the ASM unique.

The issue of ensuring co-existence of systems with heterogeneous log formats has been proposed by IBM where each log file is converted into a standard called Common Base Event (CBE). In the real time working environment converting every log file into CBE is a time consuming process and is therefore a bottleneck.

2.1 Common Base Events (CBE):

CBE [10][11] is the standard format to represent the log file of the products and is proposed by IBM in the year 2001.

The log messages generated by two products are ever the same, and there is enormous diversity in the style and format of messages between applications. Also, message logs are invariably product-centric, adhering to standards and terminology that are unique to a particular vendor (or even to a particular application). Under these circumstances, there is need for consistency of interpretation and for an analysis tool to understand an application's messages. These messages have to be in an expected form, using expected terminology. An open standard that fits this requirement is the Common Base Event model. This model provides a basis for problem determination and a cornerstone of Autonomic computing system management. Furthermore, Common Base Events are XML structure that contains XML messages targeted for use in a Web services environment which opens the possibilities for autonomic management across products, even those from different vendors.

3. PROPOSED WORK

The proposed work is an extension of an alternate approach which was published by the authors [12][13][14][15], that allows each vendor to write their own log file in the format required for them, yet able to coordinate in heterogeneous work culture and improves the performance by avoiding writing multiple adapters for each log file of every product, as it exists with IBM's approach. The followings are the new features to the language allowing definition of complex structure of the symptoms and associated actions to realize the self managing capability of the Autonomic system.

1. Creation of nested policies using simple policies.
2. Improved creation of policies based on the product specific vocabulary and avoiding need for adapters.
3. Improved housekeeping activities like modification, deletion, searching of policies/ symptoms, registering and unregistering of managed applications and log variables.

3.1 System vocabulary

The architecture of the autonomic system is shown in the Fig.1 and Fig. 2.

3.1.1 Autonomic System Manager (ASM)

The Autonomic System Manager (ASM) is the core part in the proposed architecture. It provides the administrator with a user friendly language editor which allows user to create policies for the application. User is endowed with the ability to create complex rules by having rule expressions as combinations of various events and operations between them.

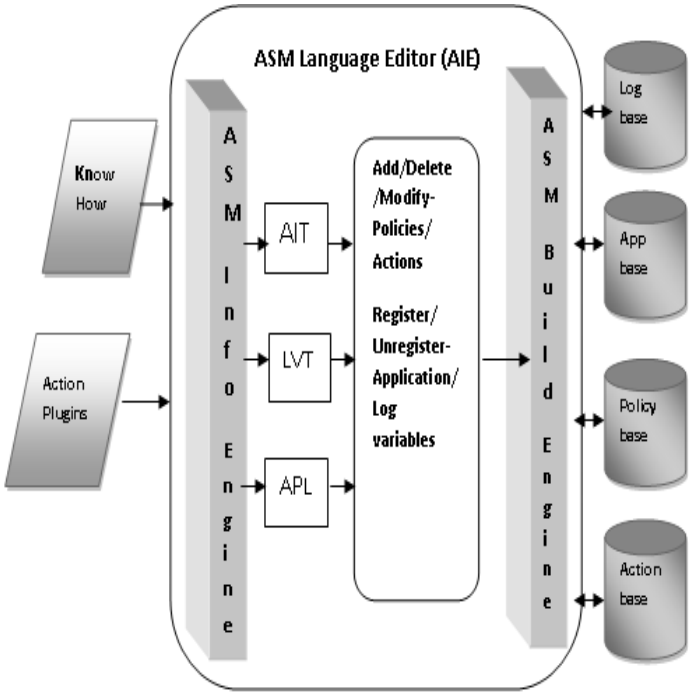

Fig. 1 ASM Language Editor

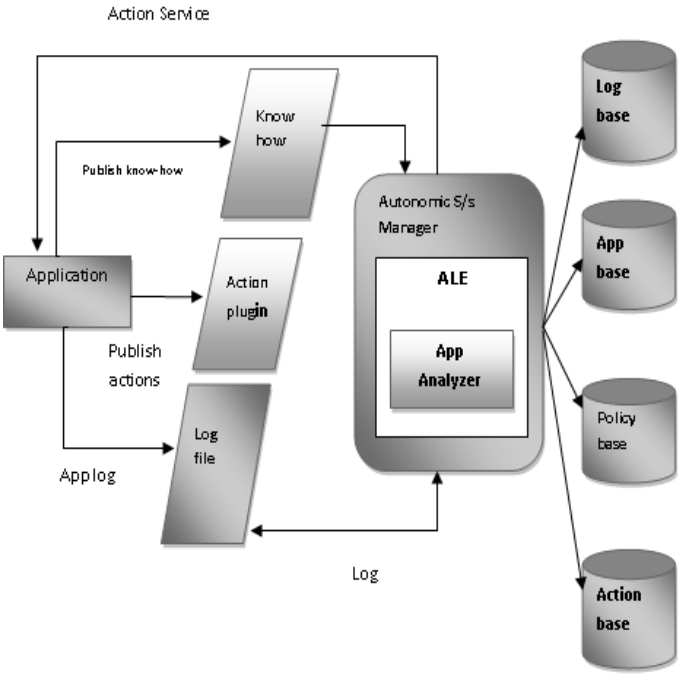

Fig.2. System Architecture

The simpler rules can be reused for the creation of complex rules called nested policies. This is the new feature of the language to create the complex rules in existing ASM.

3.1.2 Application Analyzer

Application analyzer component registers the product with Autonomic manager and analyzes the log file format of selected applications. The AA must get connected to the know-how structure of the product. Know-how provides the details of all product specific variables and their data type used to create product log file. 'LogBase' is a collection of all

such product specific variables used to construct the policies across the application. Each log variable will be identified by its application id which helps the administrator to avoid any naming conflicts in Logbase. The know-how is expressed in terms of LEX and YACC specification files.

3.1.3 ASM Language Editor

The ALE is an editor that facilitates writing new policies, editing the existing policies and deletion of policies in specialized language.. The editor has engine to process the language statements. It interacts with LogBase to know the product specific log variables, ActionBase to manage the actions, AppBase to manage applications registered and PolicyBase to manage policies.

3.1.3.1 Policy Base(Symptom Catalog)

Policy Base is the repository of symptoms and reference to action base for intended action. A symptom is the pattern describing the possible situation of interest i.e problem scenario. Each symptom describes causes and effects. Structure and an example of policy base is as follows.

Structure of PolicyBase:

Policy ID @ Policy Description # Policy Condition $ Action to be executed.

Illustration of Policy base:

polid=pol001@poldesc=desc # cs-method=GET && cs-uri-stem=/hello.html$actionid= act001 .

This policy triggers action- 1 listed in the action-base identified by ‘ actionid= act001’ when the log variable ‘cs-method ‘ is ‘GET’ and the log variable ‘cs-url-stem’ is ‘ hello.html’

3.1.3.2 Action Base

Action base is the collection of java code to be used in creating the policies. They are executed whenever the policy matches with the parameters during runtime. Structure and an example of Action base is shown below.

Structure of ActionBase:

Action Id @ Action Description # Language of Action plug in $ Location of action plug in.

Illustration of Action base:

actionid=act001@action_desc=Sending SMS# action_lang =java$action_code=SendSms.java

The action code written in java language is available in the action listing with the file name ‘SendSms.java’. This action is referred with its code i.e ‘actionid=1’ in the policy with identification ‘P01001’ which is described above.

3.1.3.3 Appbase

AppBase is a collection of information about all registered Applications that includes Application identifier, Application name, Log file location and Know How location etc. Structure and an example of Appbase is shown below.

Structure of AppBase:

ApplicationId @ ApplicationDescription # Application Name $ Knowhow location of application* Application Logfile location.

Illustration of Action base:

app_id=app001@app_desc=Web_server#app_name=iis $knowhow=c:/log.y*logfile=c:/logfile/ex001.log

IIS web server is the name of the registered application which is identified by ‘app_id=app001’, its Know-how location is ‘ c:/log.y’ and log file location is ‘c:/logfile/ex001.log’

3.1.3.4 LogBase

LogBase is repository of all log variables from various applications or managed resources. These log variables are s constructed when application is registered in autonomic manager. These registered Log Variables are used to create policies. Log variables naming conflicts is resolved using application identifiers. Structure and an example of Logbase is shown below.

Structure of LogBase:

LogVariable Id @ Application Identification # Caption of Log variable $ Log variable description

Illustration of Logbase:

logvar_id=lvr001@app_id=app001#logvar=time$logvar_desc =time_of_logfile_creation

The app_id= app001 is a registered application that contains a log variable with the name ‘time’ and identification ‘ lvr001’, and describes the time of log file creation.

3.1.4 Know-How

Know-how is a mechanism to interpret the structure of log file of the product. It provides the detailed information about the registered log variables which are used to create the policies. Know- how of each product is specified by each product vendor using the generic notations called/regular expressions/Language grammars’ used by compilers. During the runtime the location of the know-how of registered application is determined and the log variables are extracted and stored in the LogBase. As an illustration IIS log file is used and its Know-how is a LEX and YACC specification for the IIS web server.

3.1.5 Action plugin

The Action Plugin file contains the proposed action classes in target language like Java and provides actions to resolve the problem of an Application.

3.1.6 ASM Info Engine (AIE)

AIE is responsible for collecting information for further processing. It registers the application and its log file by sending request to ASM Build Engine (ABE). Once the Application is registered App Info Table (AIT) of editor will be loaded with application information and application data will be stored in AppBase. AIE component gets connected to know how file of the application. It extracts all log variables and structure from knowhow and interacts with ABE component for registering log variables along with their application identifiers. The collected log variables are deposited into LogBase by ABE. The Editor’s Log Variable Table(LVT) will be loaded with log variables. ActionPlugin

List (APL) contains all available actions to be chosen for policy writing.

3.1.7 ASM Build Engine(ABE)

ABE is responsible for validating and processing all the language statements sent from AIE. It invokes inbuilt language engine to execute the statements. ABE further converts the language statements into structured data and stores them into respective repositories.

3.2 Language Specification

The description of the language specifications to represent symptoms includes whitespaces, keywords, operators, identifiers, and statements are shown in the Table-1

Table 1. Keywords and operators

Category	Keyword/Operators	Purpose
Operators	Relational operators- <, >, <=. >=, = and <> Logical operators- AND, OR	Used to create formulate policy conditions and group the conditions
Policy /Nested policy Creation	CREAT, NEW, POLICY,POLID, POLDESC ,LOGVAR ,LOGVAL ,NESTED	Create new policy /nested policy , convert to structured format and add it to polbase
Modify/delete Policy	DELETE,MODIFY ,SET_VAL, END_VAL ,OLDVAL, NEWVAL, REM_VAL,ADD_VAL ,SET_ACT ,REM_ACT ,OPR ,ADD_ACT	Delete policy /modify stored policy from polbase and apply policy condition connector
Action Creation	ACTION, ACTIONID ACTION_DESC ,ACTION_LANG ,ACTION_CODE	Create new Aaction plugin , convert to structured format and add it to actionbase
Register application	REGISTER ,APPLICATION ,APP_ID , APP_DESC, APP_NAME, KNOWHOW	Collect necessary information for managed application and store it in appbase.
	,LOGFILE	
Register Log variable	LOGVARIABLE, LOGVAR_ID, LOGVAR , LOGVAR_DESC	Register log variable of particular application and store it in logbase
Unregister application/ log variable	UNREGISTER	Unregister application/ log variable from appbase and logbase respectively
Control Structure	IF, THEN	Used to formulate policy condition

3.3 Working of the proposed system

The proposed system is developed in Java and tested using IIS web server. The complete working of the system is shown in the sequence diagram of Fig. 3.

1. Administrator sends request for Application Register to AIE along with Application Information and KnowHow
2. AIE sends Application registration request to AppBase
3. AIE also sends Log Variables registration request to LogBase
4. Application will be registerd with AppBase
5. All log variables of registered application are registered with LogBase
6. Admin sends request to AEC to register new policy for the system
7. AEC validates the policy statement and forwards the policy to ABE for further processing
8. ABE sends request to AppBase to verify whether the Application/s mentioned in the
9. policy is/are already registered
10. AppBase verifies the Application of the policy
11. ABE sends request to LogBase to verify whether the all the log variables mentioned in the policy are already registered
12. LogBase processes ABE request for log variables
13. ABE then sends policy registration request to PolicyBase
14. ABE sends policy registration information to Admin.
15. ABE processes the statement and sends registration request to ActnBase
16. ActionBase registers the action and sends confirmation to ABE
17. Admin sends request to AEC for registration

of new action

18. AEC validation the action statement and forwards it to ABE for further processing.
19. ABE sends Action registration information to *Admin*.

3.3.1 Test Cases

Case -1 Creation of Simple Policy :

```
CREATE NEW POLICY POLID=policy_id_string POLDESC=string
{
  IF LOGVAR=string LOGVAL=string
  THEN ACTIONID= action_id_string
}
```

Purpose: Creates new simple policy to be added to policy base where all autonomic system policies are stored.

Description: The Creation of Simple policy language statement consists of two parts namely 'policy condition' and 'policy action'. The 'IF LOGVAR=string LOGVAL=string' is policy condition where LOGVAR is a name of one of the log variable of LogBase and LOGVAL is a possible value of given log variable. The 'ACTIONID= action_id_string' is policy action which describes the action to be taken in case of policy match. The value for ACTIONID must be chosen from existing actions of the system.

Example:

```
CREATE NEW POLICY POLID=pol006 POLDEC= desc
{
    IF LOGVAR=sc-status LOGVAL=402
    THEN ACTIONID= act001
}
```

The above statement creates new simple policy for Microsoft Internet Information Services 5.1 application and describes, if log entry having 'sc-status =402' is found then action 'act001' must be executed. The action 'act001' refers to a java program. The sample log entries of IIS web server that can match the statements are as follows.

```
#Software: Microsoft Internet Information Services 5.1
#Version: 1.0
#Date: 2010-07-07 10:49:14
10:49:14 127.0.0.1 GET /iisstart.asp 402 19453
```

The Fig. 4 shows the language editor. The left side of the editor window contains Log Variables Table of IIS web server. The right side of the editor window contains Action table that lists all the actions added to the system. On successful execution of policy creation statement new policy will be added to polybase (policy repository).The Fig. 5 shows addition of new policy namely 'polid=pol006@poldesc=desc#sc-tatus=404$actionid=act001' to the polyBase after the execution of above statement.

Case -2 Creation of Nested policies:

```
CREATE NESTED POLICY POLID= policy_id_string POL_DESC=string
{
  IF POLID = policy_id_string AND POLID= policy_id_string
[AND/OR POLID= policy_id_string]
    THEN ACTIONID= action_id_string
    [AND ACTIONID= policy_id_string.
     ACTIONID ]
}
```

Purpose: Creates new independent policy from the existing policies.

Description: The Creation of Nested policy language statement consists of two parts namely policy condition and policy action. The 'IF POLID = policy_id_string AND POLID= policy_id_string [AND/OR POLID= policy_id_string]' is policy condition where POLID value must be 'id' of one of the existing policies. Any number of policies can be used along with operators AND/OR to derive the new policy. The policy condition of new policy will be derived as combination of all the policy condition of given policies. Newly created policy is not dependent on the policies on which it is created. So policy deletion does not involve any cascading effect.

The 'THEN ACTIONID= action_id_string [AND ACTIONID=policy_id_string. ACTIONID] ' is a policy action which describes the action to be taken in case of policy match. The value for ACTIONID must be chosen from existing actions of the system. ACTIONID can also be derived as action of one of the existing policies which is given as ACTIONID=policy_id_string. ACTIONID i.e. id of existing policy followed by dot operator and key word ACTIONID.

Example :

```
CREATE NEW NESTED POLICY POLID=pol006 POLDESC=Nested_policy

    {   IF POLID=pol001 OR POLID=pol002
        THEN ACTIONID= pol001.ACTIONID
    }
```

The above language statement creates new policy using existing policies pol001 and pol002 as shown in Fig. 6 and Fig. 7 below. The policy condition of new policy will be policy condition of pol001 or policy condition of pol002.The action id of new policy is the same as the action id of policy pol001 as mentioned in 'THEN ACTIONID= pol001.ACTIONID'.

The policies pol001 and pol002 are as follows :

polid=pol001@poldesc=desc#cs-uri-stem=/headings.htm$actionid=act004

polid=pol002@poldesc=desc#cs-uri-stem=/hello1.htm$actionid=act002

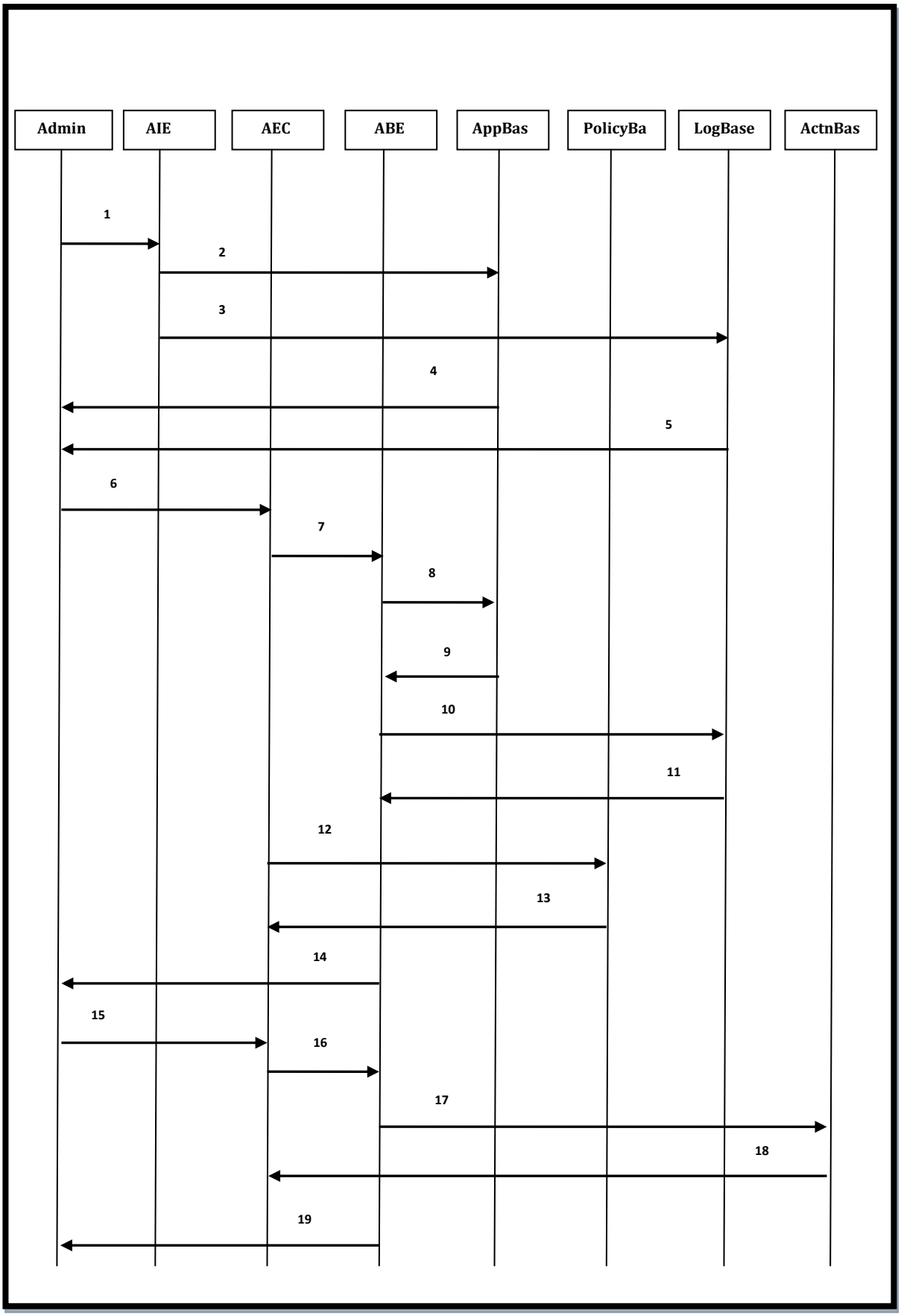

Fig. 3: Sequence Diagram

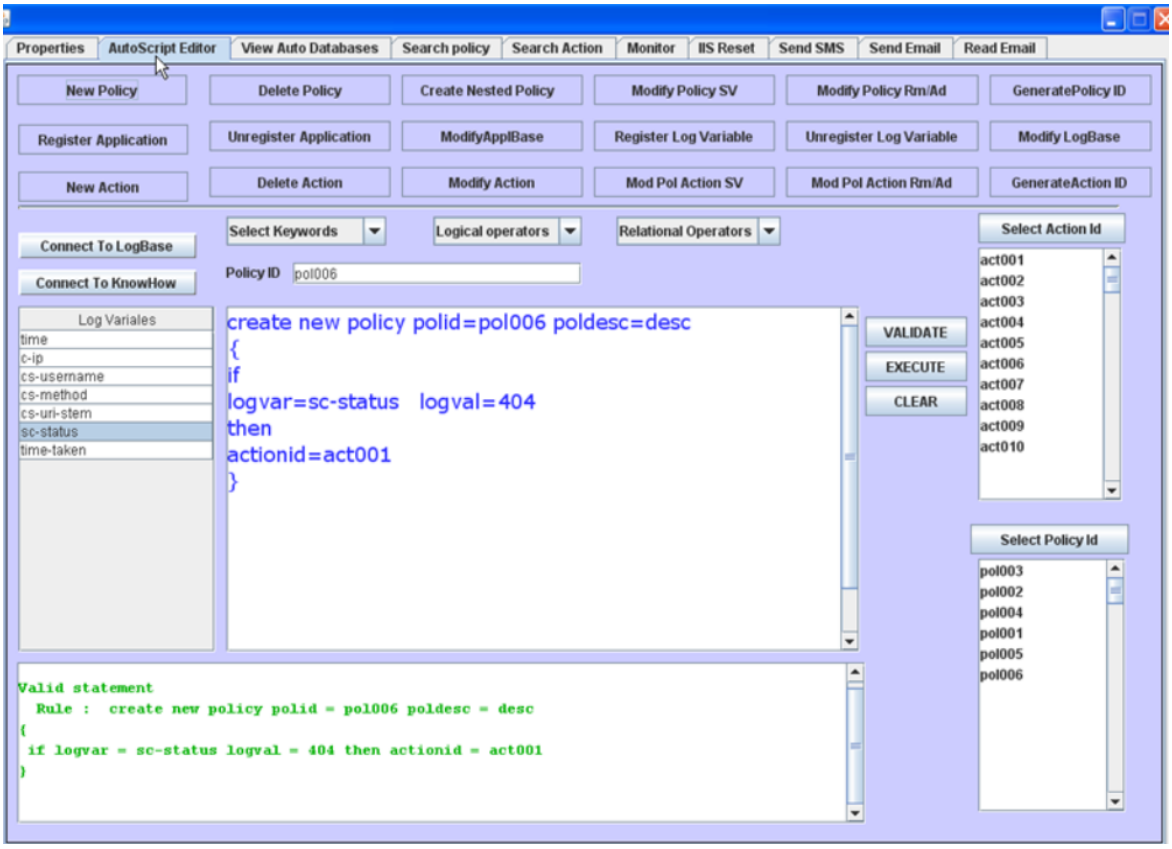

Fig. 4: Creation of simple policy

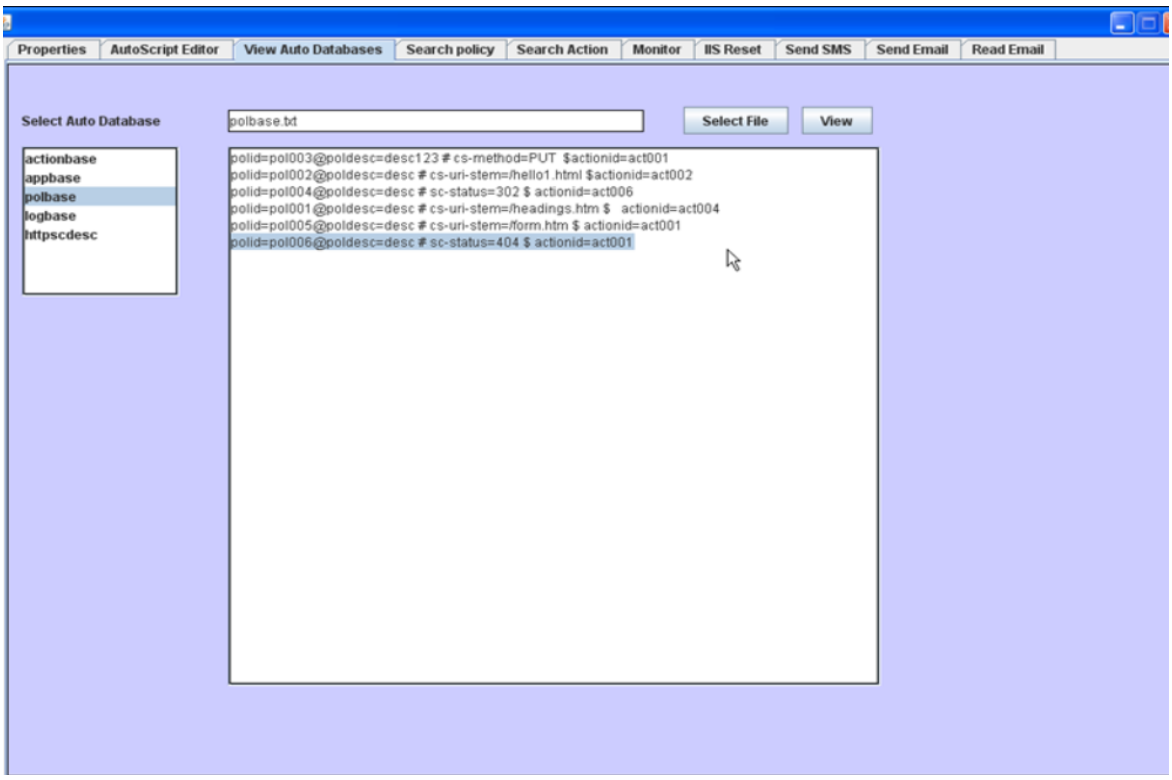

Fig 5: Polybase after creation of simple policy

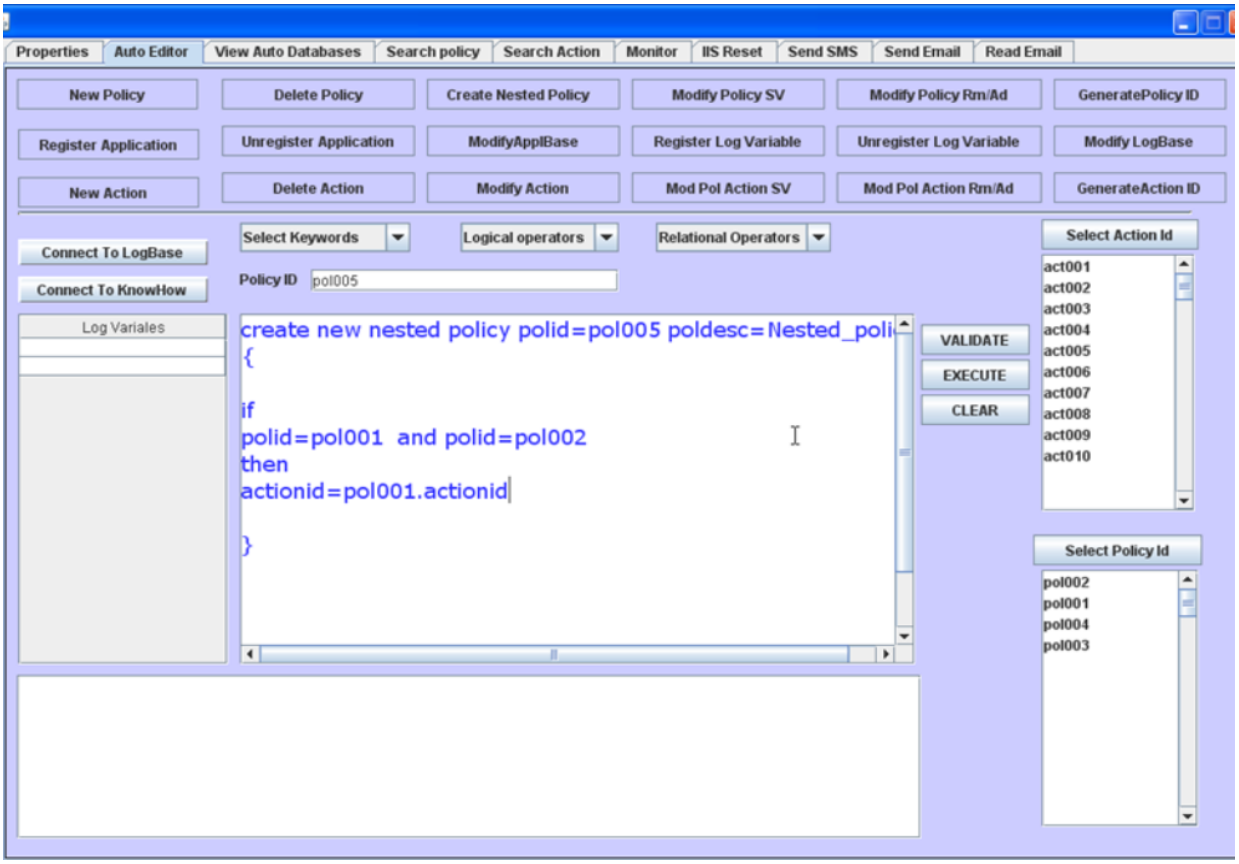

Fig.6. Creation of Nested policy

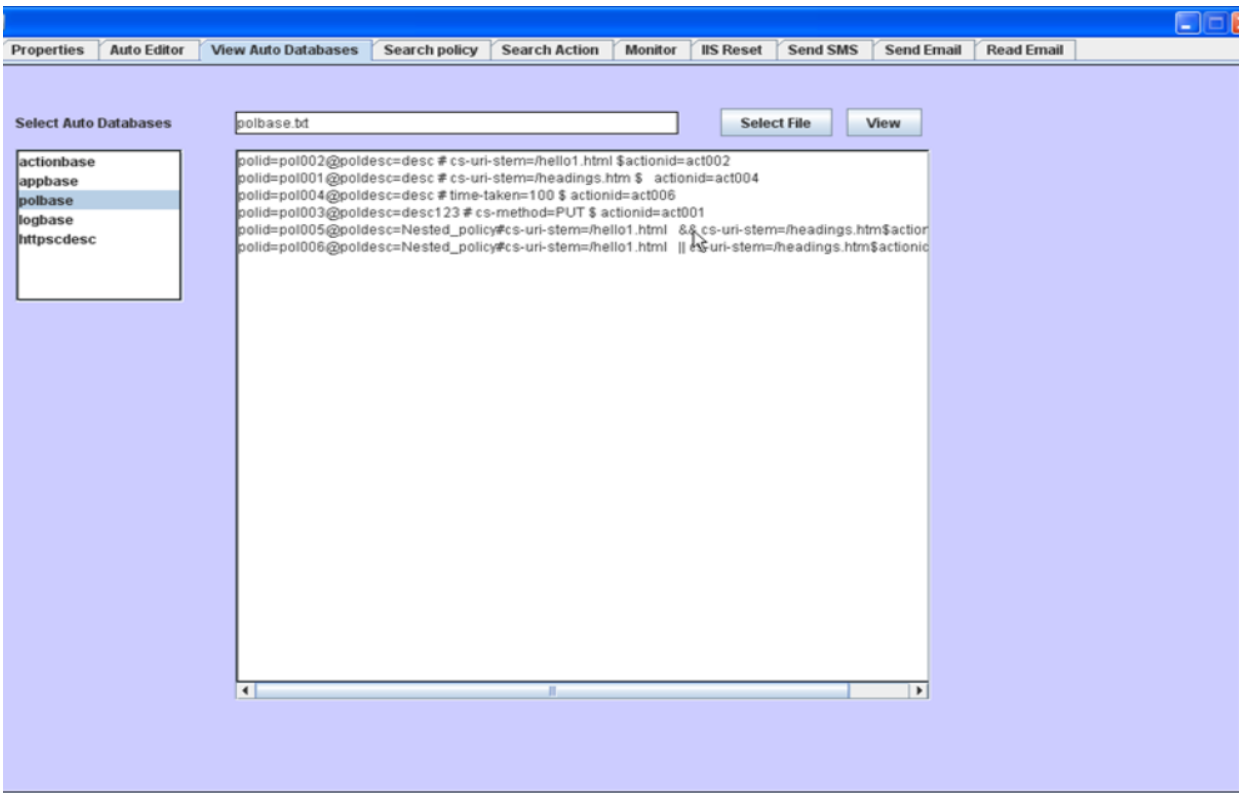

Fig. 7. PolyBase after creation of Nested policy

4. CONCLUSION

This proposed work presents a new user friendly language editor to build symptom catalog to implement Self healing mechanism with know-how in an Autonomic System. It provides a well defined language for building databases of Autonomic system such as Appbase, LogBase, PolicyBase and ActionBase .ASM language editor empowers administrator with the ability to create and maintain complex policies for any application in a convenient way. ASM language editor also presents a more user friendly way of implementing actions by letting administrators know about the errors occurring in the application. Also, the manual action execution feature (for any error occurring) presents a method by which administrator can be ensured of full control of the working of any application. ASM language editor helps to convert unstructured application data into structured information. This structured data is systematically stored into Autonomic system repositories. The systematically written policies and collected data make it easy to automate the application and find solution for recurring system problems. The system can be enhanced to handle administrators reply for taking action. The language can be modified to support creation of cross application policies using centralized Appbase and Logbase .

5. REFERENCES

[1] Jeffrey O. Kephart, David M. Chess., " The vision of Autonomic Computing ", IEEE Computer Society, January 2003, IBM Thomas J. Watson Research Centre.

[2] Paul Horn (senior vice president), " Autonomic Computing IBM's perspective on the state of Information Technology", IBM Research.

[3] Hassan, S.; Al-Jumeily, D.; Hussain, A.J. "Autonomic Computing Paradigm to Support System's development," Developments in eSystems Engineering (DESE), 2009 IEEE Second International Conference, vol., no., pp.273-278, 14-16 Dec. 2009

[4] Henxing Zhao; Congying Gao; Fu Duan; , "A survey on Autonomic computing research," Computational Intelligence 2009 and Industrial Applications, 2009. IEEE PACIIA 2009. Asia-Pacific Conference on, vol.2, no. pp.288-291, 28-29 Nov.

[5] IBM " An Architectural Blueprint for autonomic Computing ", whitepaper, June 2005.

[6] A. G. Ganek, T. A. Corbi, " The dawning of the Autonomic Computing Era ", IBM System Journal Vol. 42, No. 1, 2003.

[7] M.G.Hinchey and R. Sterritt, "Self Managing Software in Computer", IEEE Computer Society, Vol.39, Pages 107-109, Feb. 2006.

[8] S. Hariri and Et al., "The autonomy Computing Paradigm in Cluster Computing", The journal of Networks, Software tools and applications, Springer Science Business Media, B.V.(Kluwer, academic publishers), Vol. 09, Pages 5-17, Jan. 2006.

[9] E. Patouni, N. Alonistioti, "A Framework for the development of self managing and self configuring components in autonomic environments", International Symposium on a world of wireless, Mobile and Multimedia Networks, Pages 480-484, June 2006.

[10] " A Practical Guide to the IBM Autonomic Computing Toolkit", www.ibm.com/redbooks/sg24-6635.pdf.

[11] DePalma, N.; Popov, K.; Parlavantzas, N.; Brand, P.; Vlassov, V. "Tools for Architecture Based Autonomic Systems," Autonomic and Autonomous Systems, 2009. ICAS '09. IEEE Fifth International Conference on , vol., no., pp.313-320, 20-25 April 2009.

[12] Liu Wen-jie; Li Zhan-huai; , "Application of Policies in Autonomic Computing System based partitionable Server", IEEE workshop on Parallel Processing, CPPW 2007.

[13] L.M. Durham, M. Mlienkovic, P. Cayton, "A Platform Support for Autonomic Computing", IEEE International Conference on Autonomic Computing, (ICAC-06), June 2006.

[14] Dr. U.P.Kulkarni, J.V.Vadavi, M.M.Math, Dr. A.R.Yardi, "A Symptom Editor: A self Healing Autonomic System" International Journal of Computer Science and Network Security VOL. 8 No. 9, September 2008.

[15] M.M.Math, Dr. Seetha. M, Dr. U. P..Kulkarni., Dr. A.R.Yardi ulkarni,A.R.Yardi-" Generic Log Adapters-A Step towards building a Parser Based Self Healable Autonomic System ", International Journal of Recent Trends in Engineering,Vol 2, No. 3, November 2009.