Prototype of Intrusion Detection Model using UML 5.0 and Forward Engineering

Muthaiyan MADIAJAGAN, Pragya GARG
Birla Institute of Technology Pilani, Dubai Campus, Dubai, UAE
jagan@bitsdubai.com, prag.garg@gmail.com

In this paper we are using UML (Unified Modeling Language) which is the blueprint language between the programmers, analysts, and designer's for easy representation of pictures or diagrammatic notation with some textual data. Here we are using UML 5.0 to show "prototype of the Intrusion Detection Model" and by explaining it by combining various parts by drawing various UML diagrams such as Use cases and Activity diagrams and Class Diagram using which we show forward engineering using the class diagram of the IDM(Intrusion Detection Model). IDM is a device or software that works on detecting malicious activities by unauthorized users that can cause breach to the security policy within a network.
Keywords: *Intrusion, Anomaly, UML, Forward Engineering, Intrusion Detection*

1 Introduction

Intrusion is the breach of security policy of a system or a network by unauthorized personals. Protection of this vital information from malicious activities of the Hackers in the era of networking has become an important issue. Suspicious activities by these attackers can be identified either by user's behavior or by user profiling by using user models. Intrusion detection is used to trace malicious activities by these attackers. Most of these activities take place at the host machine. Maximum number of such anomalies is carried out from a host machine and they sometimes remain undetected by few network based intrusion systems.

IDS monitor the network by finding signs such as that of thwart or intrusion and produce report to Management Station [4].

The World Wide Web is becoming a vast resource of information as attackers can now strategically work on much more sophisticated attacks with the growing access to the internet. With time and experience they use their unpredictable methods for attacking, making it hard for the agent to learn its approach. Each time the attacker comes with more concrete solution and a better approach for attacking the system. They are advancing in learning the changes and limitations of the operating systems, network protocols and the software implementations of various kinds. In defense to such attacks host based solutions like IDS, antivirus software, fire walls etc. are commonly used.

Host based solutions like Intrusion Detection System, various antivirus software and firewalls are usually used for anomaly types such as virus or intrusion detection. Although, these approaches are not fully accurate, they also have limitations. Thus, there arises a need to develop newer systems to overcome the ever growing network intrusion threats.

Our main focus of this paper will be to explain how an intrusion detection model works, how packets' are passing, what kind of components is making the system, how information flow occurs by the help of UML. Using UML we will explain the IDM and later will present a java code for translation of design artifacts to a foundation of a code which will show a translation from design artifacts to a foundation of a code, which is not meant to illustrate a robust, fully developed Java program with synchronization, exception handling and so on, but only for the better understanding of the diagram [3]. We have also extended by giving the UML diagrams and the Forward Engineering using java code for various classes later in this paper [1].

The Prototype is shown in the following Figure1.

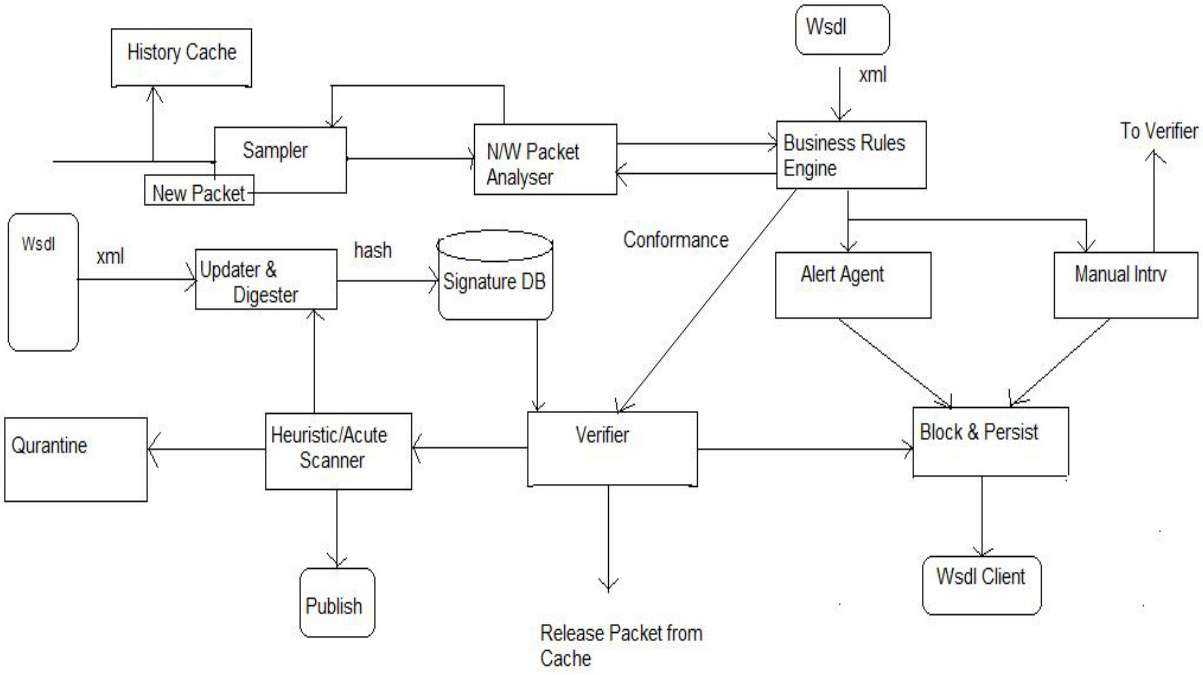

Fig. 1. Prototype of the IDM [1]

2 UML (Unified Modeling Language)

In the field of software engineering UML is a general purpose modeling language. It can be used as a visual language for specification, documentation and construction of the artifacts of the systems. The UML represents diagrammatic notation or pictures with some textual data. UML is considered a software blueprint language for analysts, designers, and programmers.UML is used for object-oriented problem solving [12].

TYPES OF UML's

There are nine types of UML diagrams which lead to simplification of the problem such as Use-cases, Class, Object, Sequence, Collaboration, State chart, Activity, Component and the Deployment diagrams.

Figure 2 shows a Use- Case UML diagram drawn in UML 5.0 version. It shows a standpoint of an external user. It emphasis more on what a system does rather than how it is done. Use-Cases are connected to scenarios. A scenario is an example of what happens when someone interacts with the system. It is showing a Use-Case relationship between Business Rules Engine, Alert Agent, Verifier and Manual Intervention. Here the Business Rules Engine will check the XML packets. It responds to the DOS attacks, Filtering, Screening and Authentication. If at all any errors are encountered then Alerts will be audited, Logged and mailed to concerned authorities along with the Alert Agent Component. If at all any anomaly is detected Business Rules Engine sends it to Manual Intervention Component as well. Administrator will manually check for errors and correct then if sent to Manual Intervention Component. For further Analysis of the XML packet Manual Intervention Component will further send it to the Verifier. The following figure is showing a Use-Case relationship between Business Rules Engine, Alert Agent, Verifier and Manual Intervention. Here the Business Rules Engine checks the XML packets. It will respond to the DOS attacks, Filtering, Screening and Authentication. If at all any errors are encountered then Alerts will be audited, Logged and mailed to concerned authorities along with the Alert Agent Component. If at all any anomaly is detected Business Rules Engine sends it to Manual Intervention Component as well. Administrator will manually check for errors and correct then if sent to Manual Intervention Component. For further Analysis of the XML packet Manual Intervention Component will further send it

to the Verifier.

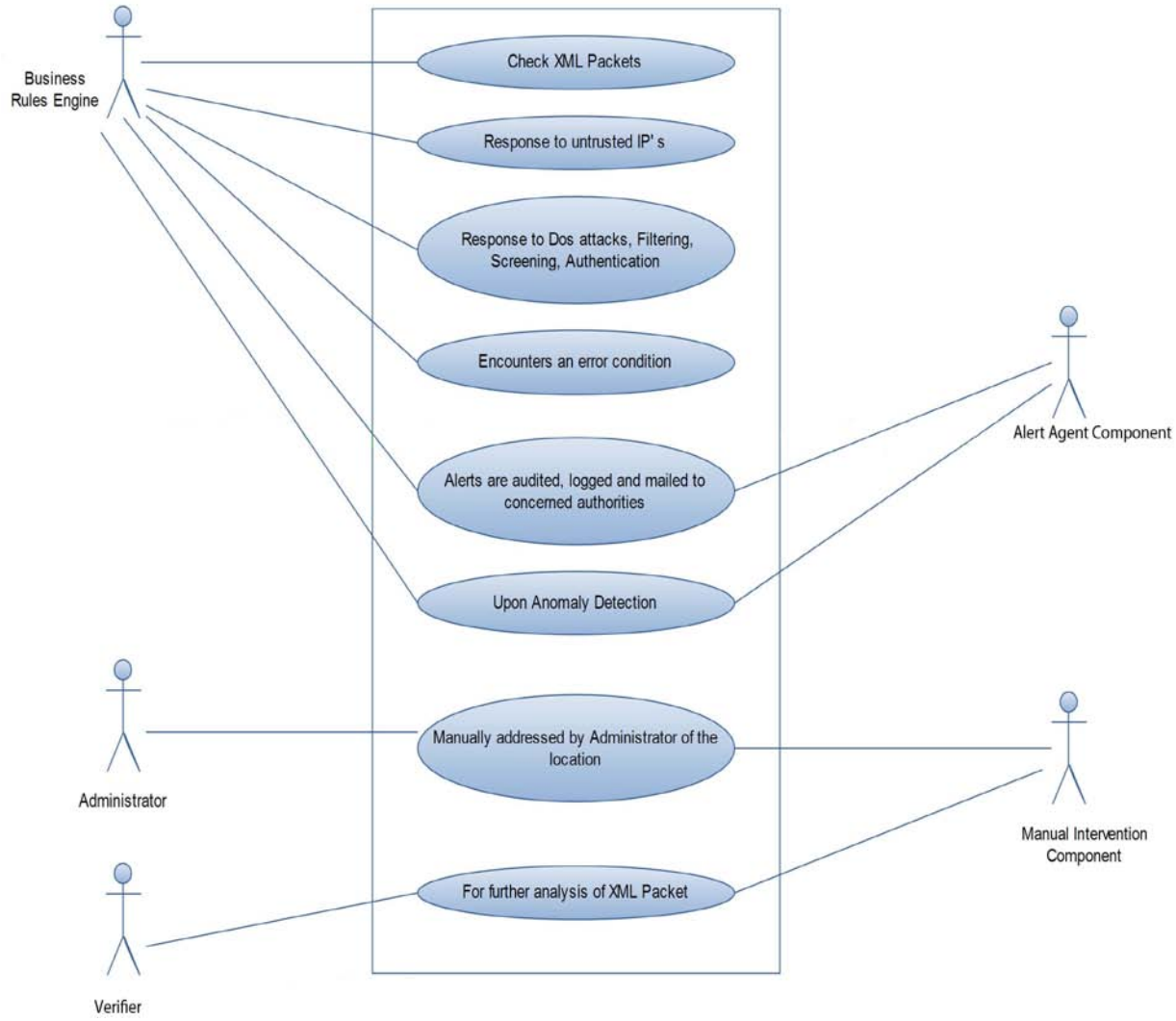

Fig. 2. Use-Case diagram drawn between Business Rules Engine, Alert Agent, Verifier and Manual Intervention using UML 5.0

Figure 3 represents the same components i.e. the Business Rules Engine, Manual Intervention, Alert Agent and Verifier in UML 5.0 using Activity Diagram. Activity diagram focuses on the flow of activities involved in a single process. The Activity Diagrams shows how these activities depend upon one another. Activity Diagrams can be divided into object swim-lanes which will determine which object is responsible for which activity. A single transition will come out of each activity connecting it to the next activity. A transition branches into two or more mutually exclusive transitions. Guard Expressions (inside []) label the transitions coming out of a branch. A transition may fork into two or more parallel activities, which combine later in the form of solid bars. The diagram begins with a start circle at the beginning and ends with concentric black/white stop circles towards the end. The activities are rounded circles. In this figure the diagram starts with a black circle marking the beginning of the activity diagram. As soon as the anomaly is detected, for each error is either sent to the Alert Agent Component or the Manual Intervention. If sent to the Alert Agent Component then the Alerts are audited, logged and is mailed to the concerned Authorities. If sent to the Manual Intervention component if it succeeds then it goes to the manually

addressing to the administrator of the location. Upon failure will go for further analysis to the verifier. In order to check for known attack signatures send the documents to the verifier, this activation diagram ends with concentric black/white stop circles towards the end.

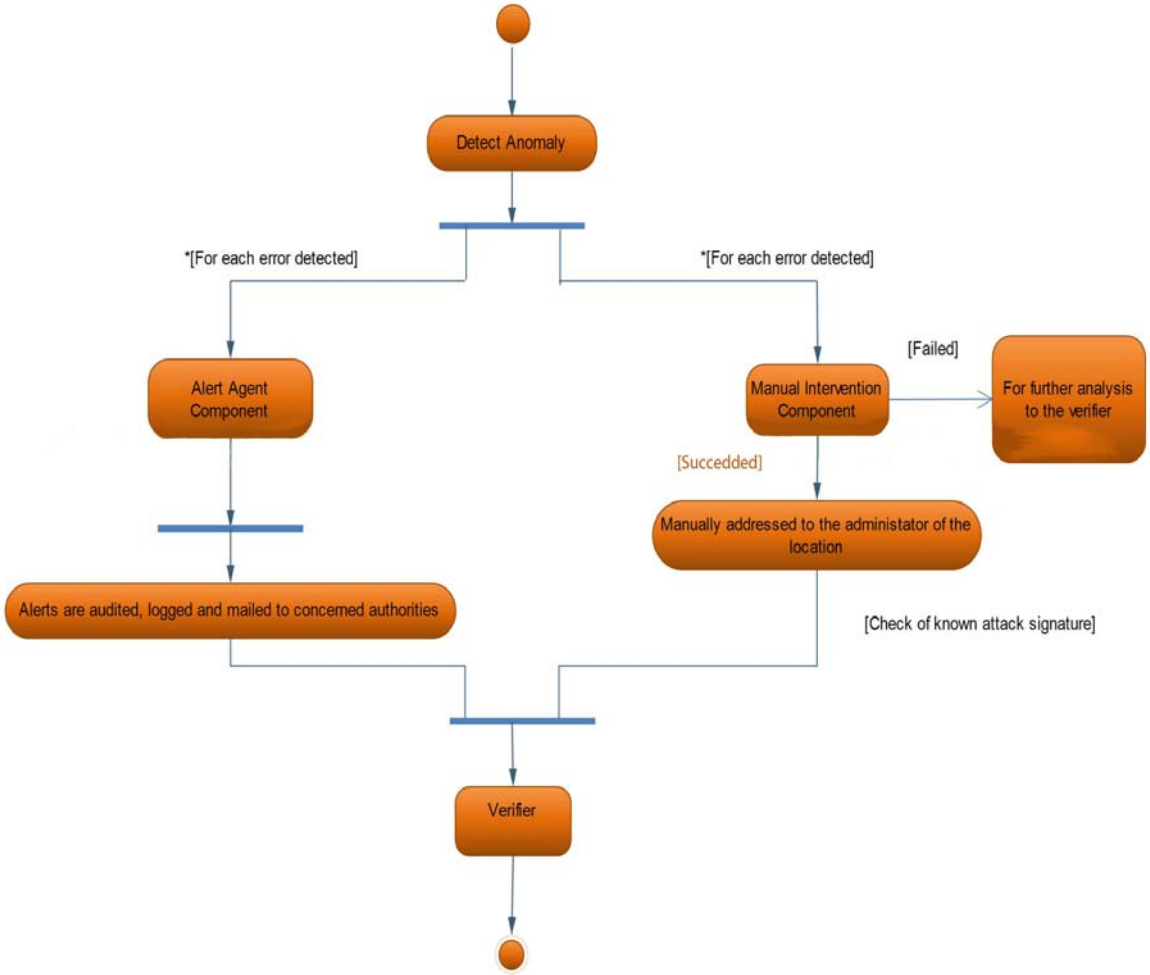

Fig. 3. Activity Diagram between the Business Rules Engine, Manual Intervention, Alert Agent and Verifier using UML 5.0

Figure 4 is showing an Activity Diagram of Database, Updater and Verifier. Here the activation diagram begins with a start circle at the beginning. Local Database is used. Database is used to send web services based notices to all instances. Updater receives all the web services updates and keep updating, changing, upgrading the database with new information. Using Hash Algorithm picked up XML packet with given payload are digested. The verifier checks all the packets against attacks.

Finally attack is prevented and activity diagram ends with concentric black/white stop circles towards the end.

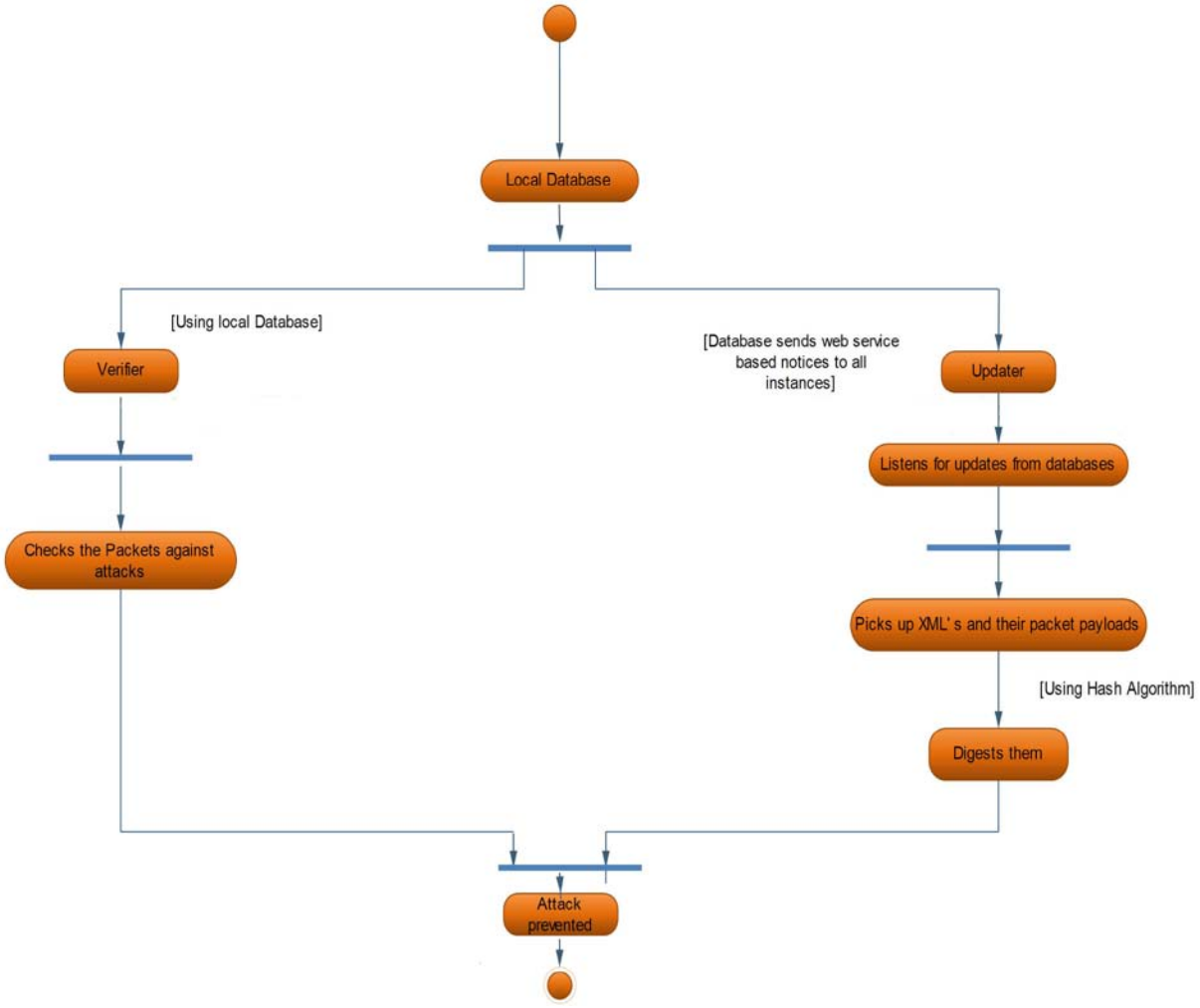

Fig. 4. Activity Diagram between Database, Updater and Verifier using UML 5.0

Figure 5 shows a Use-Case diagram of components and actors involved are the Cache memory, Sampler, Business Rules Engine, Network Packet Analyzer and the Pre-processing Engine. Here Cache Memory collects network packets.

It can either be a cache hit/cache miss. Sampler will heuristically pick up sample network packets, and send them to the Network Packet Analyzer. Network Packet Analyzer and Pre-Processing Engine will analyze the packets and convert them into XML documents. They will go to the Business Rules Engine which will check XML documents for anomalies, detects un-trusted IP' s, detects screening, Dos attacks, filtering and screening.

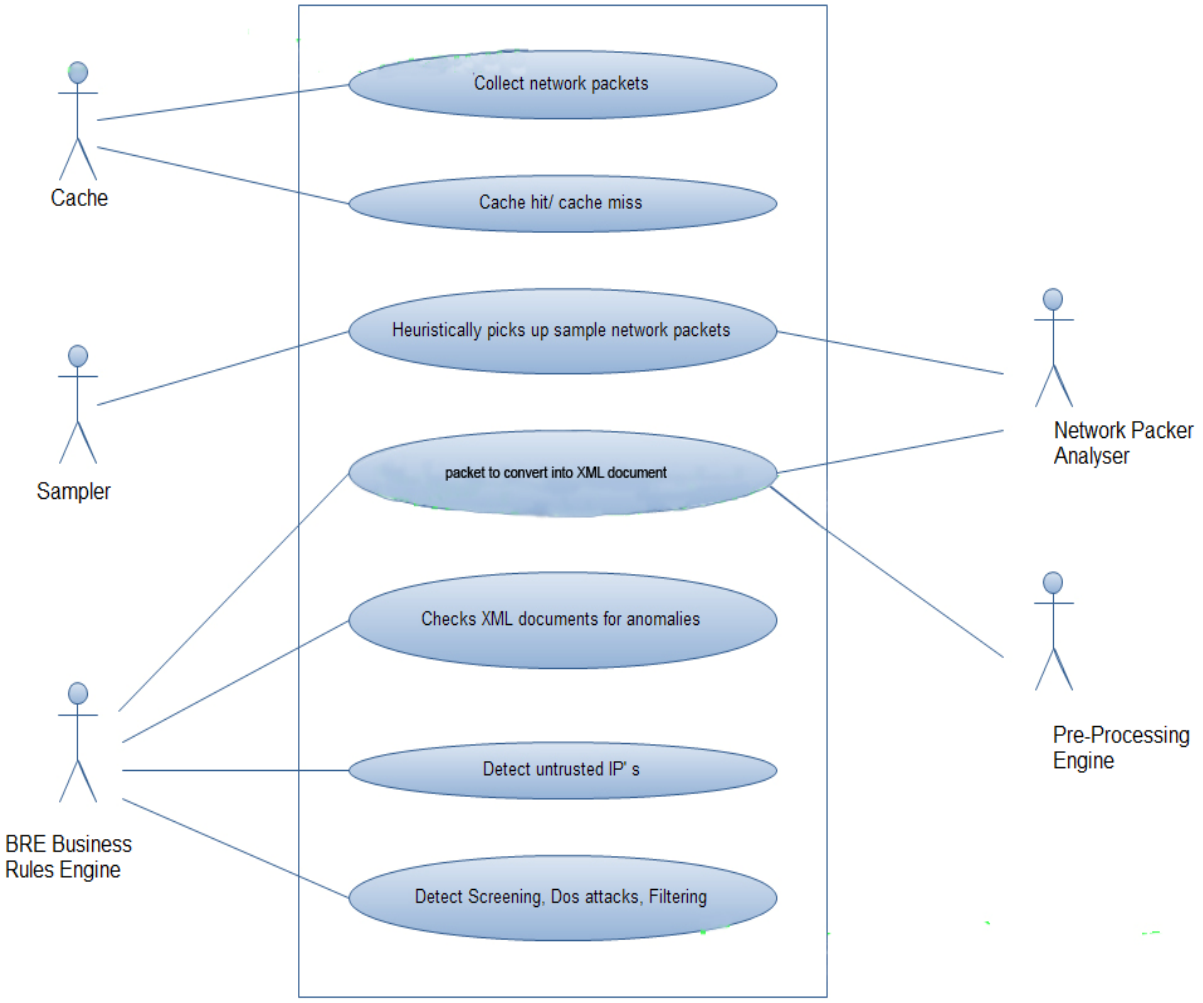

Fig. 5. Use-Case diagram of components and actors involved are the Cache memory, Sampler, Business Rules Engine, Network Packet Analyzer and the Pre-processing Engine using UML 5.0

Figure 6 shows a class diagram covering all components like the Sampler, Alert Agent, Database, Verifier, Updater, Manual Intervention and the Business Rules Engine. A class diagram is used to give the overview of a system with its classes. Class diagrams are static-they are only used for displaying what interacts, how the components interact is not shown. UML Class diagram consists of a class name, attributes, and the operations. Class diagrams have three kinds of relationships: Association-It is a relationship in between instances of two classes. Aggregation-It shows the collection of an entire class. It is shown by a diamond end pointing to the part containing the whole and Generalization-It shows the class-super class relationship. Class diagrams can show various multiplicities such as 0...1(zero or one instance), 0...* or *(either none, or no limit on the number of instances), 1(exactly one instance), 1...*(at least one instance) [4].

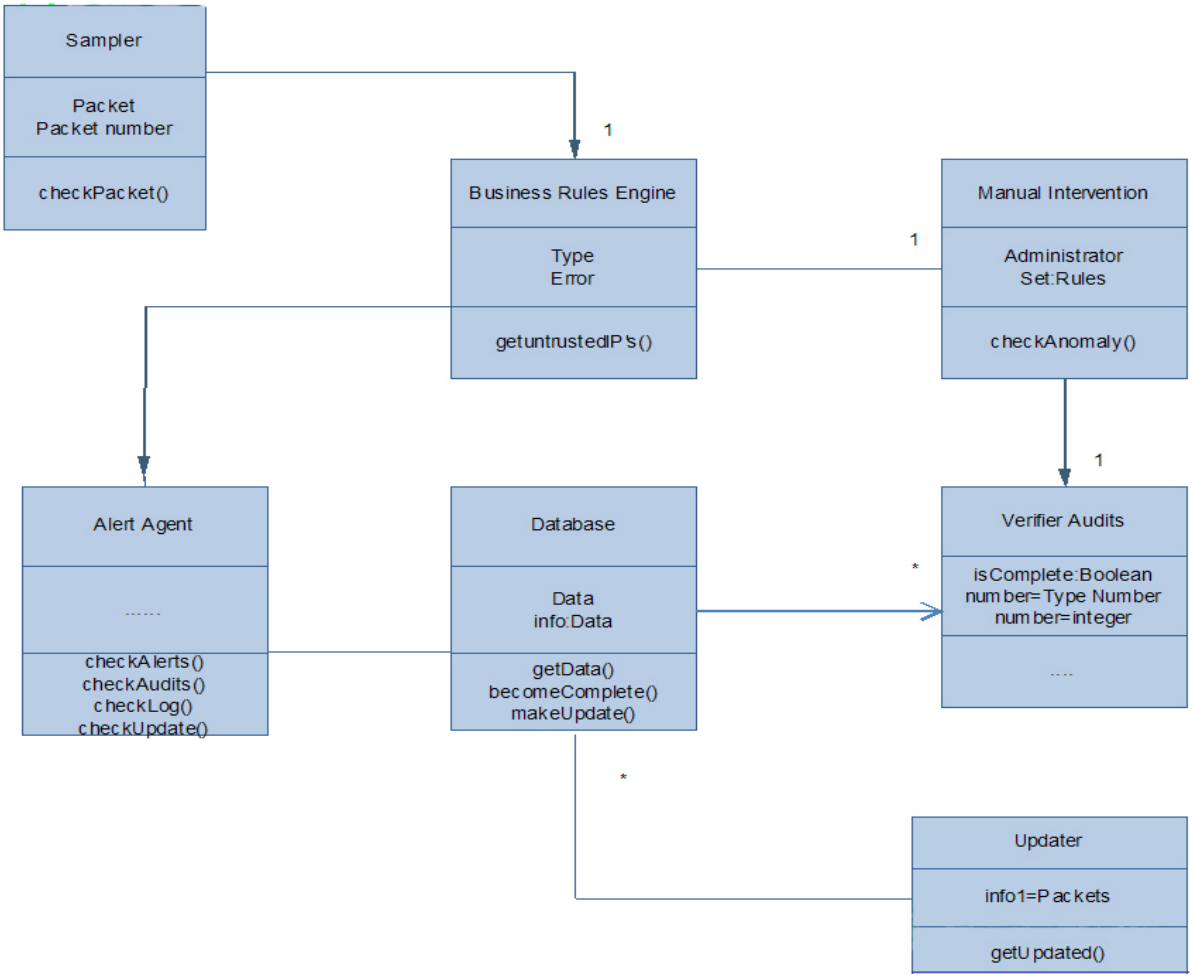

Fig. 6. Class –Diagram using UML 5.0

3 Java coding

This section will present a domain layer of the classes in Java for this Iteration. The main point here is that there is a translation from design artifacts i.e. from a UML class diagram drawn in UML 5.0 version to a foundation of code. This code is not meant to illustrate a fully developed, robust Java Program with synchronization and exception handling [3].

Figure 6 represents a Java Program for IDM Prototype.

```
      Class Updater
Public class updater
{
Private Packet info1;
Public        Updater        (Packet
changeoccured)       {info1       =
changeoccured ;}
public  Packet  getinfo1  ()  {return
info1; }
}
```

```
      Class Business Rules Engine
Public class Business Rules Engine
{
private Map<ItemRules,
ManualIntervention>
       Interventions = new
HashMap()<ItemRules,
ManualIntervention>;

public Business Rules Engine()
{
ItemRules  ir1  =  new  ItemRules(  100
);
ItemRules  ir2  =  new  ItemRules(  200
);
        Rules set = new Rules( 30 );
   Manual Intervention Inter;
        Inter     =     new     Manual
Intervention(  ir1,  set,  "packet  1"
);
interventions.put( ir1, inter);
inter  =  new  ManualIntervention  (
ir2, set, "packet 2" );
interventions.put( ir2, inter);
```

```
}
Public Manual Intervention get Manual Intervention( Item rules ir)
{
returninterventions.get( ir );
    }
}
      Class Alert Agent
public class Alert Agent
{
privateBuisness Rules Engine Engine;
public Alert Agent( Business Rules Engine Engine );
{
this.Engine = Engine;
}
public void checkAlerts()
{
currentAlerts.becomeComplete();
}
public void getAudited( Item rules ir, int number );
{
Manual intervention inter = Engine.getManualIntervention(ir );
}
public void get logged()
{
current log = new log();
}
public void getUpdate ( changeoccured );
     }
}
      Class Manual Intervention
public class Manual Intervention
{
private Item ir;
private Rules set;
private string intervention;
public Manual Intervention
    (ItemRulesir, Rules set, string intervention )
}
publicItemIRgetItem() { return ir; }
public Rules getRules() { return rules; }
public string getIntervention() { return intervention; }

Class Database
public Class Database
{
private List<VerifierAudits> Audits = newArrayList()<VerifierAudits> ;
private Number number = new number();
privatebooleam is complete = false;
private Updater updater;
public packet getUpdated()
{
return updated.getInfo1(). minus(getData() );
}
public void become complete () { isComplete = true; }
public void makeAudits
( ManualIntervention Inter, int number)
{
Audits.add( new verifier Audits( inter, number));
public packet getData()
{ Packet data = new Packets();
       Packet subdata = null;
for( Verifier Audits Audits = Audits )
{
subdata = Audits.get subdata();
Data.add( subdata );
}
return data;
}
public void make update( Packet changeoccured)
{
update = new update( changeoccured );
}
}

Class Verifier Audits
public class verifier Audits
{
privateint number;
private Manual Intervention intervention;
public verifier( Manual Intervention Inter, int number )
{
this.intervention = inter;
this.number = number;
}
public packet get subdata()
{
returnIntervention.get value().times( mumber );
}
}
Class Sampler
Public class Sampler
{
privateBuisness Rules Engine Engine = new Buisness Rules Engine Engine();
private Alert Agent alert agent = new Alert Agent( Engine );
public Alert Agent get Alert Agent()
{return alert agent; }
```

4 Conclusion

The proposed architecture will manage the distributed system components efficiently. It will allow new computing resources and services to be added dynamically. Most of the challenges faced by current IDS are addressed by the proposed architecture. We have successfully explained and simplified the Prototype of the IDM using blueprint language UML, version 5.0. We have combined various components and actors for various for Class, Activity and Use-Case diagrams. We have also shown Forward Engineering with the help of Class diagram using java code.

Future Work

Reading specifications from a file and drawing the diagram using program. Create a UML diagram through a program in JAVA/VB i.e. Reverse Engineering which is totally opposite of what we have done in this paper. We aim at getting UML diagrams directly from Java or some other language with different diagrams apart from class diagrams like sequence diagrams.

References

[1] International Journal of Recent Trends in Engineering, Vol. 1, No. 1, May 2009.

[2] International Journal of Recent Trends in Engineering, Vol. 1, No. 2, May 2009.

[3] C. Larman, *An Introduction to object-Oriented Analysis and Design and Iterative Development*.

[4] S. R. Pressman, *Software Engineering, A Practioner' s Approach*.

[5] P. China and Huangshan, "Proceedings of the Second Symposium International computer Science and Computational Technology (ISCSCT' 09)" 26-28, Dec. 2009, pp. 134-138.

[6] M. Chapple and E. Tittle, *Certified information systems security professional*.

[7] International Journal of Recent Trends in Engineering, Vol. 2, No. 2, November 2009 (Nanyang Technology University), RESEARCH PAPER.

[8] International Journal of Recent Trends in Engineering, Issue. 1, Vol. 1, May 2009, RESEARCH PAPER.

[9] http://technet.microsoft.com/en-us/library/cc751219.aspx (via. Google.com)

[10] A. Schwartzbard, A.K. Ghosh, *A study in the Feasibility of Performing Host-based Anomaly Detection on Windows NT*.

[11] M. Speciner, C. Kaufman and R. Pearlman, "Network Security".

[12] Ontology for Host-based Anomaly Detection-Margareth P. Adaa (Oslo University College) May 23, 2007.

[13] R. King and G. Govanus "Windows 2000 Network Security Design".

Madiajagan MUTHAIYAN holds a M.S., in Software Systems from BITS, Pilani, India and a PhD in Component based software Development. He has 15 years of College / University teaching experience and 2 years of experience in Blue Chip Software Company. Presently, he is working as Senior Lecturer, CS, BITS, Pilani-Dubai. His areas of interest include Component Based Software Engineering, Distributed Database Systems, Software Architecture, and Theory of Computation. He is a Professional member of Professional bodies ACM, World Enformatica Society and Computer Society of India.

Pragya GARG is presently a final year student in B.E Computer Science. She has experience in IT section and is currently working at GBM (IBM) in Dubai as a Software Sales Trainee in the year 2011. She has Technical skills in C, Java, JavaScript, PHP, HTML, SQL and UML.