REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
19-02-2004	Final Report	01-Dec-00 - 19-Feb-04

4. TITLE AND SUBTITLE

Formal Methods for Information Protection Technology
Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks
Part I

5a. CONTRACT NUMBER
ISTC Registration No: 1994p

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Professor I.V. Kotenko Ph.D.

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

St. Petersburg Institute For Informatics & Automation of the Russian Academy of Sciences
39, 14th Liniya
St. Petersburg 199178
Russia

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0014

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ISTC 00-7035

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited. (approval given by local Public Affairs Office)

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking St. Petersburg Institute For Informatics & Automation of the Russian Academy of Sciences as follows: Formal Methods for Information Protection Technology
The use of open computer networks as an environment for exchange of information across the globe in distributed applications requires improved security measures on the network, in particular, to information resources used in applications. Integrity, confidentiality and availability of the network resources must be assured. To detect and suppress different types of computer unauthorized intrusions, modern network security systems (NSS) must be armed with various protection means and be able to accumulate experience in order to increase its ability to front against known types of intrusions, and to learn new types of intrusions. The project will perform three main tasks.
1. Develop a mathematical model and a tool that simulates various coordinated intrusion scenarios against computer networks;
2. Develop the mathematical foundations, architecture, and principles of implementation of autonomous-software-tool technology implementing the learning system for intrusion detection;
3. Develop the fundamentals, architecture and software for the computer security system based on multi-level encoding for information protection in mass application.
Currently, scientific efforts in network security area are undertaken mainly in the development of the network defense mechanisms. Unfortunately, substantially less attention is paid to the study of the nature of intrusions and, in particular, remote distributed intrusion attempts. No appropriate tools for intrusion/attack simulation nor research on a formal framework for intrusion specification exists.

TASK 2
To detect and suppress different types of computer intrusions, modern NSS must be able to accumulate experience in order to increase its ability to front against known type of attacks/intrusions and to learn unknown simple and complex, local and distributed types of attacks. This requires the use of a powerful intelligent learning subsystem (LS) in NSS. That is why the second task of the project concerns to the development of the formal model, architecture, and software prototype of the autonomous intelligent learning system for detection of the attacks/intrusions against computer network. Two main focuses are the core of this research. The first one is a decomposition of the whole learning task into multitude of sub-tasks according to the ontology of attacks/intrusions and allocating them among specialized autonomous learning software systems/modules. The second focus is a mathematical issue of the generic autonomous learning abilities of a software module. It is necessary to select or to develop a number of mathematical methods to cover the necessary and sufficient functionalities of generic learning software modules to cope with the learning over the data of various formats. One more mathematical issue of this research is to develop a multi-level interaction among the learning software modules of various levels realizing meta-classification idea within multi-

member knowledge discovery from data architecture applied to the attack/intrusion detection LS.

15. SUBJECT TERMS

EOARD, Mathematical & Computer Sciences, Computer Systems

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			/Signed/PAUL LOSIEWICZ, Ph. D.
UNCLAS	UNCLAS	UNCLAS	UL		**19b. TELEPHONE NUMBER** *(Include area code)* +44 20 7514 4474

ST. PETERSBURG INSTITUTE
FOR INFORMATICS AND AUTOMATION

(SPIIRAS)

EUROPEAN OFFICE OF AEROSPACE
RESEARCH AND DEVELOPMENT

(EOARD)

Project #1994 P

Formal Methods for Information Protection Technology

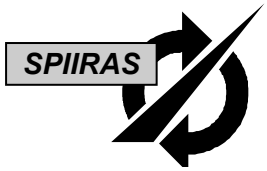
Final Report

Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks

Part I

Principal Investigator of Task 2
Leading Researcher of the Intelligent
Systems Laboratory of SPIIRAS
Ph.D. Professor I.V. Kotenko

St. Petersburg
November 2003



ST. PETERSBURG INSTITUTE
FOR INFORMATICS AND AUTOMATION
(SPIIRAS)

EUROPEAN OFFICE OF AEROSPACE
RESEARCH AND DEVELOPMENT
(EOARD)

Final Report

Project # 1994P
Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks

Part I

Principal Investigator of Task 2
Leading Researcher of the Intelligent
Systems Laboratory of SPIIRAS
Ph.D. Professor I.V. Kotenko

St. Petersburg

November 2003

Contents

Preface

This volume is the Final Report on *Task 2 of the Project #1994P "Formal Methods for Information Protection Technology".* The Project is being performed according to the agreement between European Office of Aerospace Research and Development (EOARD), The International Science and Technology Center (ISTC) and St. Petersburg Institute for Informatics and Automation (SPIIRAS).

The title of the Task 2 of the Project #1994P is *"Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks".*

This Project is devoted to exploration of the basic principles of the intrusion detection learning (learning data sources, learning data representation and processing, basic algorithms and structure of learning), multi-agent architecture of intrusion detection learning system and software implementation of its basic components.

According to the Work Plan, during the period from the Project beginning (on December 1, 2000) three interim reports summarizing intermediate results were submitted that are

1. Interim Report #1 on the Project 1994P, Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks. *Interim Report #1,* SPIIRAS, August 2001 [InterRep#1],
2. Interim Report #2 on the Project 1994P, Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks. *Interim Report #2*, SPIIRAS, May 2002 [InterRep#2], and
3. Interim Report #3 on the Project 1994P, Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks. *Interim Report #3*, SPIIRAS, May 2003 [InterRep#3].

The intermediate results were also presented in the workshop "Novel Information Technologies and Information Assurance" organized by Air Force Research Laboratory (Information Directorate), European Office of Aerospace Research and Development and St. Petersburg Institute for Informatics and Automation hosted by Binghamton University (March 4-7, 2002).

All theoretical results and conclusions of the research are explored and validated via simulation on the basis of the software developed by authors. The developed software can be demonstrated in AFRL/IT as well as software code can be submitted to the Partner on demand.

Nine papers describing results of the Project have been accepted for presentation and publication in Proceedings of International Conferences (including publication in *Lecture Notes in Computer Science* and *Artificial Intelligence* series), one paper has been published in *The International Journal of Computer Systems Science & Engineering* (see section "Publication of the Project results").

According to the Work Plan the following tasks have been solved:

B-1. Development of the learning task ontology, allocation of learning tasks over generic learning agents.

B-2. Development of an architecture of the Multi-agent Learning System and mathematical methods realizing learning functionalities of the generic agents.

B-3. Development of the protocols of inter-level intelligent agent interaction (negotiation), generalization of the particular agent decisions according to the meta-classification approach and development of architecture of the Multi-agent Learning System as a whole.

B-4. Development of object-oriented conceptual project of the Multi-agent Learning System.

B-5. Development of the software prototype of the Multi-agent Learning System implementing theoretical results of research and its evaluation.

B-6. Evaluation of the properties, advantages and disadvantages of the developed architecture and mathematical methods implemented within the prototype of the intelligent Multi-agent Learning system.

All the tasks provided by the Work Program are solved completely.

The interim reports presented the basic results of the research implied by Work Plan. To express the final view of the Project results and to make this Report in some sense self-contained, it is written in the way that summarizes all main results of the research obtained during three year period with the focus on the implementation issues and simulation results, which have not yet been highlighted in the previously

submitted interim reports because these issues were the main subject of the research efforts during the last three quarters.

Demonstration of the developed technology and supporting software tool can be performed on request. The software itself can also be supplied on request.

Thus, all the researches provided by the Task 2 of the Project 1994P are successfully carried out according to the Work Plan. This Report outlines them as a whole. The main results and proposals for future research are given in the final Report conclusion.

Principal Investigator of Task 2
Leading Researcher of the Intelligent Systems Laboratory of
the St. Petersburg Institute for Informatics and Automation
of the Russian Academy of Sciences

Ph.D. Prof. Igor Kotenko

Report Summary

This Report presents the final view of the Project results. It is written in the way that summarizes all the results of the research obtained during three year period with the focus on the implementation issues and simulation results, which have not yet been highlighted in the previously submitted interim reports because these issues were the main subject of the research efforts during last two quarters.

Intrusion detection learning (IDL) is a task of distributed data processing aiming at engineering knowledge bases and decision making mechanisms responsible for detection of illegitimate operations of network and computer users. The main peculiarity of this learning task for the purposes of computer network assurance system as compared with conventional data mining and knowledge discovery from data is that in the former case it is necessary to take into account several heterogeneous information sources. At that each such a source only partially specifies an abnormal user activity or attack against computer. This is why the detection of the above abnormalities in usage of a computer can be successful only if the detection technology is organized on the basis of several data sources because each of them contains only a fragment of information about attack "traces" and the complete "picture" can be assembled on the basis of combining these fragments or decisions made on their basis.

This Project is focused on the *development of Intrusion Detection Learning Systems (IDLS) components based on use of data fusion principles and built as multi-agent system*. In other words, the research focus is development and prototyping of a software infrastructure supporting collaborative semi-automated work of specialists in design and implementation of applied IDS, in particular, in design and implementation of its decision making components.

The research results described in the Report are presented in five chapters and two appendixes. A brief summary of the Report contents is given below.

Chapter 1 "Peculiarities of Intrusion Detection Learning Task. Methodology and Models of Intrusion Detection Learning". The Chapter introduces into the problems to be solved and outlines peculiarities of the intrusion detection learning task as well as the basic ideas that form the milestones of the onward research. It outlines the main peculiarities of intrusion detection learning task and presents a brief overview of the modern days research on this subject. It also introduces generally the approach accepted in this Project. The Chapter considers the main concepts of logging and auditing of the events, happening in computer networks to be defended, presents analysis of the data structures and examples of audit data used in modern operating systems, security systems, and applications. The Chapter analyses data sources, which can be used for training and testing of the IDLS. The IDLS data sources taxonomies, classifying data sources due to location of source and software generating data, processing level, and an object associated with data, are presented. The typical structures, measurement scales, size and dimensionalities of the data used for the IDLS training and testing are analyzed.

Also it reviews the proposed decisions regarding to the structure of the learning data and their usage in attack detection and learning. In general, this Chapter aims to make it clearer the peculiarities of data to be mined, the kind of diversity of this data structures, specificity of the distributed data mining and knowledge discovery within the framework of this particular task. One of the project objectives is investigation of the feasibility and advantages of use of both multi-agent and data and information fusion technologies for intrusion detection learning task. This Chapter gives motivations of the potentially important advantages of use multi-agent paradigm.

The Chapter is also considers the basic aspects of the methodology of multi-agent intrusion detection learning adopted within this Project. Particularly, it discusses the existing principles of intrusion detection learning and motivates the choice adopted, analyzes the potential structures of distributed decision making and decision combining ("Decision fusion meta-model"). The Chapter describes the developed structure of intrusion detection learning system distributed knowledge base, structure of the distributed decision making mechanisms and outlines their interactions with the intrusion detection learning ontology. It gives short information concerning particular techniques used in this Project for engineering of the local knowledge bases, local solvers, and particular techniques used.

The Chapter specifies mathematical methods of data mining and knowledge discovery that are available for use in the developed prototype of IDLS. The methods described in this Chapter are divided into five groups: (1) Methods for combining decisions of multiple classifiers; (2) Methods for mining numerical (continuous) data; (3) Methods for mining discrete data; (4) Methods for mining frequent patterns and association rules from transactional data; (5) Methods for mining temporal data.

All the considered methods for combining decisions divided into three groups: (1) Voting methods; (2) Probability-based or fuzzy methods; (3) Meta-learning methods based on stacked generalization (meta-classification methods); (4) Meta-learning methods based on classifiers' competence evaluation. A modified approach to competence-based classifier combining is developed. Two types of methods are selected and

implemented in IDLS: meta-classification methods and methods based on competence evaluation. These methods are outlined in more detail. The developed temporal mining algorithm based on statistical properties of the temporal vector-wise sequences of binary and/or numerical data is considered.

A certain attention is also paid to the training and testing methodology with accent on peculiarities entailed by distributed nature of training and testing data sources and also methodology of allocation and management of training and testing datasets, which possesses a number of specific features caused by the interdependences and constraints imposed on admissible splitting of the entire learning dataset into training and testing ones and also on admissible allocation of their parts to particular solvers to be trained.

All the above aspects of intrusion detection learning system methodology are considered within the context of the known approaches and each choice is motivated. The methodology proposed in the Chapter constitutes the basis for multi-agent architecture of intrusion detection learning system, development, implementation and deployment within a computer network supported by the developed software tool that are the subjects of the subsequent chapters of the Report.

Chapter 2 "Intrusion Detection Learning System Design Implementation and Deployment Issues. Ontology of Intrusion Detection Learning". The Chapter presents thorough mainly conceptual description of the developed and implemented technology of multi-agent information fusion system engineering intended for design, implementation and deployment of applied multi-agent data and information fusion systems. Intrusion detection system is considered in the Project as a particular application of the general data and information fusion problem. At the beginning, the Chapter outlines general view of multi-agent data and information fusion system technology and divides the engineering processes into two classes, which are those supporting design, implementation and deployment of the reusable components of the multi-agent system under design and those supporting design and implementation of data and information fusion–oriented functionalities. Respectively, two software tools used for support of the engineering processes of each of the above two classes developed by the Project authors are described. The first of them called Multi-Agent System Development Kit, MASDK, supports design procedures of the first, mostly of general purpose, class of technological processes while the second one, Information Fusion Design Toolkit, mainly supports design procedures of the second class, i.e. those specifically destined for making use in design of data and information fusion–oriented functionalities. Description of these toolkits, description of the content and peculiarities of design procedures carried out by each of them are the main subjects of this chapter.

The Chapter describes conceptually the developed model of "Generic agent" that is considered as a basic component supporting technology for applied software agent design and implementation. "Generic agent" comprises reusable components forming in some sense an "empty" agent, which after specialization (according to an application to be designed) is transformed in semi-automated mode into instances of an applied software agent of IDLS. The Chapter also presents the technology and its implementation issues concerning the semi-automated specialization of "generic agent" resulting in particular software agent of IDLS.

The Chapter also describes the developed ontology specifying a high-level representation of the basic notions of Intrusion Detection Learning domain. According to the modern view of the information system technology, ontology is one of the most important components of every information system, in particular, if such an information system is highly distributed, is of large scale and knowledge-based. It forms the high-level conceptual model of the basic shared knowledge represented in the form of the structured multitude of basic notions with clearly and undoubtedly defined semantics independent of accepted design and implementation issues of a particular application. Ontology is presented in the system software implementation as a very important part of the software, which plays the role of a factor providing the developed software by integrity and consistency. In a multi-agent system (including the system under development) it provides much easier mutual understanding of different agents operating in distributed manner with lack of much knowledge about environment and other agents. The specific of the subject domain under research is that it combines knowledge and therefore, combines ontologies, from different domains, namely, "*Data Fusion and Data Fusion Learning problem domain ontology", "Intrusion Detection application ontology"* and *"Intrusion Detection Learning application ontology"*. These ontologies are developed and described in this Chapter.

Chapter 3 "Multi-agent Architecture and Operation of Intrusion Detection Learning System". The Charter is devoted to the conceptual view of the architectural and technological issues of IDLS design and implementation. It formulates the main assumptions accepted in the design of IDLS that concretize interaction of IDLS components and target IDS. The Chapter presents the developed multi-agent architectures of Multi-agent IDS and IDLS and discusses some important issues that are specific for intrusion detection learning. The proposed IDLS architecture comprises two types of components: (1) data sources-based learning components responsible for knowledge discovery from particular datasets and (2) a meta-level component responsible for learning meta-level classifier aiming at combining decisions of source-based classifiers. According to this respect, the Chapter describes generic tasks resulting from the entire task decomposition and allocation these tasks over particular

agent classes. It outlines also some peculiarities of the multi-agent and data and information fusion technologies that are used in design and software implementation of IDLS components.

The conceptual view of the structure of agent communication is outlined. Three types protocols needed for support of agent interactions are considered: (1) protocols that support agent message exchange; (2) protocols aiming at management of semantically interconnected dialogs (conversations) of agents; (3) protocols supporting cooperative work of agents in distributed design and learning procedures.

General configuration of generalized IDS based on information fusion (IF) approach and the structure of its distributed knowledge base are described. It is suggested that IDS includes the agents of the following classes: Information Fusion (Decision combining) management agent for brevity called hereinafter System Managing agent (SM-agent); Agent-classifier of meta-level called hereinafter for brevity Meta-Classifier agents (MC-agents); Source-based Base Classifiers (BC-agents); Data source managing agents (DSM-agents). The standard scenario of the IDS operation is outlined.

Intrusion Detection Learning Scenario is suggested. It includes engineering of the shared component of the application ontology, design of the binary classification tree, design of the meta-model of decision combining (decision tree), engineering of base classifiers and meta–classifiers, testing of IDS as a whole and monitoring of the learning process. These phases of intrusion detection learning scenario are described.

Chapter 4 "Case study Description". The Chapter specifies the developed case study that is currently used for design and implementation of the software prototype of the components of IDLS. It determines the categories and instances of attacks used in the case study, data sources and data structures representing data of the selected sources, the instances of data structures, and the examples of training and testing data.

Chapter 5 "Description of software prototypes of Intrusion Detection Learning Components. Evaluation of the developed architecture and mathematical methods implemented within the prototypes". The objective the Chapter is to present the implemented components of IDLS and some simulation results that in more details are given in Appendix.

Report conclusion presents the main results of the Project as a whole and authors' view of the further research and development.

Appendixes demonstrate the developed multi-agent technology destined for distributed intrusion detection learning and decision making.

In the end of the Report the list of the main *publications* of the results obtained within the Project is given.

The results presented in this report contain the solution of all the tasks provided by the Project.

Table of Abbreviation Used in the Report

Abbreviation	Full Text
BC	Base classifier
DF	Decision Fusion
DMT	Decision making tree
DSN	Data Source Name
DS	Data source
DSM	Data Source Management (agent)
EOARD	European Office of Aerospace Research and Development
ID	Intrusion Detection
IDL	Intrusion Detection Learning
IDL MAS	Multi-agent Intrusion Detection Learning System
IDLS	Intrusion Detection Learning Systems
IF	data and Information Fusion
IF MAS	Multi-agent Data and Information Fusion System
IFL MAS	Multi-Agent Information Fusion System
IFLS	Information Fusion Learning System
KB	Knowledge Base
KDD	Knowledge Discovery from Databases
KQML	Knowledge Query Manipulation Language
LAN	Local Area Network
MAS	Multi-Agent System
MASDK	Multi-agent System Development Kit
MC	Meta–classification (agent)
ODBC	Open Data base Connectivity
RDF	Resource Definition Language
RMI	Remote Method Invocation
SM	System manager (Information Fusion management agent)
SPIIRAS	St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences
SQL	Standard Query Language
UML	Unified Modeling Language
VAM	Visual Analytical Mining
XML	eXtended Mark-up Language

Chapter 1. Peculiarities of Intrusion Detection Learning Task. Methodology and Models of Intrusion Detection Learning

Abstract. The Chapter briefly formulates the main lessons learnt from contemporary studies on data mining for intrusion detection and IDLS prototyping, analyses the main notions regarding the audit data used for intrusion detection and intrusion detection learning and describes the methodology developed for design of multi-agent IDLS.

The approach accepted in this Project to intrusion detection system design is data centric in the sense that historical interpreted audit data is considered as the main source of knowledge needed for intrusion detection. Chapter analyzes the main concepts of logging and auditing of the events happening in the defended computer networks, presents analysis of the data structures and examples of audit data used in modern operating systems, security systems, and applications. In general, the results of this Chapter make it clear the peculiarities of data to be mined, the diversity and heterogeneity of available data structures and specificity of the distributed data mining and knowledge discovery in the framework of intrusion detection task.

This Chapter is also devoted to the description of the basic aspects of the methodology of the IDLS operation adopted within this Project. Particularly, it discusses the existing principles of data and information fusion and motivates the choice adopted, analyzes the potential structures of distributed decision making and decision combining ("Decision fusion meta-model"). The Chapter considers the developed structure of IDLS distributed knowledge base (KB), structure of the distributed decision making mechanisms and outlines their interactions with the IDLS ontology. A developed method for mining temporal sequential data is described in detail. A certain attention is paid to the training and testing methodology with accent on peculiarities entailed by distributed nature of training and testing and also methodology of allocation and management of training and testing datasets. All the above aspects of IDLS methodology are considered within the context of the known approaches and each choice is motivated. The methodology proposed in the Chapter constitutes the basis for both multi-agent architecture of IDLS and IDLS engineering, implementation and deployment supported by the developed software tool that are the subjects of the subsequent chapters of the Report.

1.1. Introduction

At present there is no necessity to advocate much the importance of the research and development in the area of computer network assurance. It is well known that this problem is now of great concerns because of worldwide spread of computer-based technologies, which are vital components of the modern world society.

There exist many threads for computer networks, their resources and respective information systems, and malefactors permanently invent new ones quicker than counter side is able to develop an adequate response. The modern-days competition between malefactors, on the one hand, and computer network security researchers and engineers, on the other hand, is still resulting in favor of the former by many reasons. Possibly, the most important one among them is that modern computer network assurance systems use mostly manually and "ad hoc" built security systems aimed at defense against known types of attacks and other threats. The current state-of-the art in the area of computer network assurance systems forces researchers and developers to reconsider this approach. In particular, it forces the researchers and developers to focus on the development of such security systems that "would be capable to learn detection of new attacks and counter-measures in a semi-automatic mode in order to eliminate, as much as possible, the manual and ad-hoc elements from the process of building an intrusion detection system" [Lee-98].

Intrusion detection learning problem is destined to provide Intrusion Detection System (IDS) designers with automated tools and techniques for analysis of raw and preprocessed data ("audit data") and for extraction of useful attack patterns.

Nowadays this problem is recognized as a top priority one in the area of computer network and information security. It is considered as a very promising alternative for current state of the practice, which mostly relies on human expertise in identification and specification of specific patterns peculiar

for particular types of known attacks. The latter also possesses many other well known drawbacks and in practice requires formidable efforts of many high quality experts.

During the last decade this problem has received substantial studies ([Denning-87], [Teng *et al*-90], [Javitz *et al*-93], [Forrest *et al*-94], [Brodley *et al*-96], [Forrest *et al*-96], [D'haeseleer *et al*-96], [D'haeseleer *et al*-97], [Forrest *et al*-97a], [Forrest *et al*-97b], [Lane *et al*-97a], [Lane *et al*-97b], [Lane *et al*-97c], [Lee *et al*-97], [Stolfo *et al*-97a], [Stolfo *et al*-97b], [Stolfo *et al*-97c], [Endler-98], [Lane-98a], [Lane-98b], [Lane *et al*-98a], [Lane *et al*-98b], [Ghosh *et al*-98], [Hofmeyr *et al*-98], [Lee *et al*-98a], [Lee *et al*-98b], [Chan *et al*-99], [Ghosh *et al*-99a], [Ghosh *et al*-99b], [Ghosh *et al*-99c], [Hofmeyr *et al*-99], [Warrender *et al*-99], [Lane-99], [Lane *et al*-99], [Lee *et al*-99a], [Lee *et al*-99b], [Lee *et al*-99c], [Lee-99], [Luo-99], [Manganaris *et al*-99], [Prodromidis *et al*-99a], [Prodromidis *et al*-99b], [Prodromidis-99], [Clifton *et al*-00], [Eskin-00], [Eskin *et al*-00], [Fan *et al*-00], [Lee *et al*-00a], [Lee *et al*-00b], [Lee *et al*-00c], Lee *et al*-00d], [Lee *et al*-00e], [Manganaris *et al*-00], [Mukkamala *et al*-00], [Portnoy-00], [Stolfo *et al*-00], [Ye-00], [Zhang *et al*-00], [Barbara *et al*-01a], [Barbara *et al*-01b], [Bloedorn *et al*-01a], [Bloedorn *et al*-01b], [Cabrera *et al*-01], [Eskin *et al*-01], [Fan *et al*-01], [Julisch-01], [Lee *et al*-01a], [Lee *et al*-01b], [Mahoney *et al*-01], [Portnoy *et al*-01], [Schultz *et al*-01a], [Schultz *et al*-01b], [Ye *et al*-01], [Apap *et al*-02], [Dokas *et al*-02], [Eskin *et al*-02], [Hellerstein *et al*-02], [Honig *et al*-02], [Gomez *et al*-02a], [Gomez *et al*-02a], [Julisch-02], [Julisch *et al*-02], [Lazarevic *et al*-02], [Liao *et al*-02], [Mahoney *et al*-02], [Sequeira *et al*-02], [Ye *et al*-02a], [Ye *et al*-02b], [Chan *et al*-03], [Heller *et al*-03], [Hershkop *et al*-03], [Gomez *et al*-03], [Kim *et al*-03], [Lazarevic *et al*-03a], [Lazarevic *et al*-03b], [Mahoney-03], [Mahoney *et al*-03a], [Mahoney *et al*-03b], [Michael-03], [Robertson *et al*-03], [Stolfo *et al*-03a], [Stolfo *et al*-03b], [Stolfo *et al*-03c], [Stolfo *et al*-03d], [Ye *et al*-03], [Wang *et al*-03], etc.).

The most prominent works on data mining for intrusion detection have been conducted in the following research groups:

- Columbia University (S. Stolfo, E. Eskin, etc.),
- University of New Mexico (S. Forrest, P. D'haeseleer, S. A. Hofmeyr, etc.),
- University of Memphis (D. Dasgupta, J. Gomez, etc.),
- Purdue University (T. Lane and C. E. Brodley),
- Reliable Software Technologies (A.K. Ghosh, A. Schwartzbard, M. Schatz, etc.),
- University of Minnesota (V. Kumar, P. Dokas, L. Ertoz, A. Lazarevic, etc.),
- North Carolina State University (W. Lee, etc.),
- Florida Institute of Technology (P. Chan, M. Mahoney, etc.),
- George Mason University (S. Jajodia, D. Barbara, N. Wu, etc),
- Arizona State University (Nong Ye, etc.), etc.

Of course, this list is not exhaustive.

The most known approaches are twofold. *The first* of them is being developed by group headed by Prof. S.Stolfo (Columbia University). This group takes a data-centric viewpoint. *The second group of approaches* is based on the ideas borrowed from immunology. The respective models of intrusion detection learning are known as computer immunology models. Several groups are investigating such kind of models. Among them, the first most known group is headed by S.Forrest (University of New Mexico, USA) and second one is headed by D.Dasgupta (Memphis University, USA).

The basic peculiarity of the *approach developing by group of S.Stolfo* is that in it intrusion detection learning is considered as a data analysis process ([Lee *et al*-98a], [Lee *et al*-99a], [Lee *et al*-99b], [Lee *et al*-00b]).

Anomaly detection is understood as finding the normal usage patterns from the historical audit data, and contrasting them with the patterns discovered from current connections (on-line audit data). A remarkable distinction between the normal usage patterns and patterns of current connection is considered as an evidence of abnormal activity.

Misuse detection is considered as encoding and matching the intrusion patterns using the audit data. These patterns play the role of arguments of rules used for intrusion detection. The central theme of this approach is to apply data mining and knowledge discovery from database techniques to audit data represented in terms of sequential records of system calls or *TCPdump* in order to construct concise and accurate classifiers for detection and classification of anomalies.

Three basic data mining and knowledge discovery algorithms are used for learning intrusion detection classifiers ([Lee *et al*-00b], [Prodromidis *et al*-99b]). They are the (1) association rules mining, (2) frequent episodes mining and (3) rule extraction algorithms. The first and the second algorithms are used to extract useful associations and serial frequent episodes from historical sequences of audit records, whereas the third one is applied to build knowledge-based classifier. To solve the aforementioned tasks, the authors use modifications of the respective standard algorithms. The modifications aim at increasing efficiency of the existing data-flow mining algorithms due to the peculiarities of the structure and other characteristics of the audit data.

One more specific approach is based on so-called meta-classification [Prodromidis *et al*-99b]. Conceptually, the proposed approach uses several simple ("light") interacting learners instead of one complex ("heavy") one. Learning system is composed of a number of simple classifiers learned inductively on the basis of their own training and testing data. Each such a classifier is called "*base classifier*". It must be computationally efficient, though not very accurate. All base classifiers are tested on a new audit data, and each testing record is submitted to all base classifiers. While performing classification of the same audit record, each base classifier makes its own classification decision. All decisions of base classifiers are joined in a new record supplemented by the correct interpretation. These records form data for training the meta-classifier. Base classifiers and meta-classifier interact during solving the intrusion detection tasks in the same way.

The developed approach is being implemented in the framework of the JAM project – "Java Agent for Meta-learning". In this project, in order to meet the challenges of both efficient learning (mining) and real-time detection, the authors of the approach under description proposed an agent-based architecture for intrusion detection systems. This architecture supposes that the learning agents continuously compute and provide the updated (detection) models to the responsible intrusion detection agents. According to [Lee *et al*-98a], the developed framework "consists of a set of environment-independent guidelines and programs that can assist a system administrator or security officer to

- select appropriate system features from audit data to build models for intrusion detection;
- architect a hierarchical detector system from component detectors;
- update and deploy new detection systems as needed."

Among other researchers working in the field of intrusion detection learning, this group is undoubted leader because it is developing a strict formal framework for this task solution and, in parallel, takes care about implementation issues of the developing approach.

An alternative (the second) approach to the design and development of the modern learnable intrusion detection systems is being developing in the framework of so-called "*computer immunology*". The respective results can be found in ([Forrest *et al*-96], [D'haeseleer *et al*-96], [Forrest *et al*-97a], [Somayaji *et al*-98], [Hofmeyr *et al*-99], [Dasgupta *et al*-01], [Skormin *et al*-01], etc.). This approach aims to take advantages from the close analogy between the tasks of the computer network intrusion detection system and human immune system. The idea of computer immunology is to use biological principles of human immune system used for distinction between "self" and "non-self" for formalizing processes of intrusion detection in computer network. This approach seems to be very perspective one but now it is at the stage of fundamental research and is exploring usefulness of implementation of some simple basic principles within computer security task.

The main lessons learnt from these studies and prototyping of Intrusion Detection Learning Systems (IDLS) can briefly be formulated as follows:

1. *Data Mining and Knowledge Discovery from Databases (KDD) approaches* form a very promising theoretical basis for intrusion detection learning, but existing approaches and techniques cannot completely meet the needs of this field. That is why the necessity to develop particular approaches and techniques focused on peculiarities of data sources that can potentially be used for intrusion detection is remaining an important task within the problem in question. To our opinion, the main drawback of the existing approaches to available data mining is low attention to the *temporal aspects* of the above data.

2. IDSs have to operate on the basis of processing of data collected from many heterogeneous and distributed data sources and this is why it has to be considered as a particular *case of data fusion (DF) systems* [Bass-00]. Learning such systems presents a number of challenges that are the subjects of the research in distributed data mining scope, at that some of such challenges remain open. Unless the understanding of this fact, there is lack of researches which practically follow this paradigm.
3. Abnormal activity, in particular, an attack has to be considered as a *rare event* [Lazarevic *et al*-03a]. Classification of rare events entails a number of particular features determining appropriate selection of metrics used for evaluation of the classification quality and also learning and classification algorithms. Neglect of the above fact should lead to failing of the quality of IDS performance and to increase of frequency of false alarms. On the other hand, acceptance of this view results in admissibility of the assumption that the probability of occurrence of two such events within a small time interval is equal to zero.
4. A critical problem of intrusion detection and intrusion detection learning is *feature selection* ([Lee-99], [Lee *et al*-00a], [Stolfo *et al*-00], [Dokas *et al*-02], [Lazarevic *et al*-03a], [Guyon *et al*-03], [Bekkerman *et al*-03], [Bengio *et al*-03]). There is no definite viewpoint how to select the features from raw data or what kind of features have to be formed as a result of raw data preprocessing. It is clear that due to very diversity of software installed in particular computer networks (OS, applications, etc) and particular requirements to the degree of computer network security this problem has no unique solution. Analysis of the state of the art in this topic shows that the most of researches ignores features specifying *temporal aspects* although recognizes the necessity of their use for intrusion detection and pays not enough attention to usage of sequential-like features.
5. The main attention of the researchers is paid to the *anomaly detection task* and the developed techniques mostly oriented respectively. At the same time, if one deals with multiple classes intrusion detection task and reduce it to the *sequence of binary classification tasks* using respective classification tree then anomaly detection techniques can also be used is some misuse detection tasks.

Our research is focused on the development of IDLS components based on use of *data fusion* principles and built as a *multi-agent system*. In other words, the research focus is development and prototyping of a software infrastructure supporting collaborative semi-automated work of specialists in design and implementation of applied IDS, in particular, in design and implementation of its decision making components.

Multi-sensor data and information fusion (IF) is able to provide an advantageous framework for intrusion detection systems of the next generation aiming at defending not a particular hosts but computer network as a whole.

Data and information fusion is an area of information technology that aims at making decisions on the basis of joint processing of available data and information obtained from different sources distributed in space and time. The modern understanding of IF is mostly associated with a class of large scale distributed decision making tasks that is primarily referred to IF specific features of methodological kind. For example, [IF] presents such an opinion regarding comprehension of IF: “Information Fusion, in the context of its use by the Society, encompasses the theory, techniques and tools conceived and employed for exploiting the synergy in the information acquired from multiple sources (sensor, databases, information gathered by human, etc.) such that the resulting decision or action is in some sense better (qualitatively or quantitatively, in terms of accuracy, robustness and etc.) than would be possible if any of these sources were used individually without such synergy exploitation”.

As applied to the military applications, the IF problem is formulated by US Air Force Research Laboratory in terms of the most complicate and most significant military application as follows [IF]: “Information Fusion: Events, activities and movements will be correlated and analyzed as they occur in time and space, to determine the location, identity and status of individual objects (equipment and units), to assess the situation, to qualitatively and quantitatively determine threats and to detect

patterns in activity that reveal intent or capability. Specific technologies are required to refine, direct and manage the information fusion capabilities."

It is emphasized in [Bass-00] that "A significant challenge remains for IDSs designers to combine data and information from numerous heterogeneous agents (and managers) into a coherent process, which can be used to evaluate the security of cyberspace". And data fusion learning is the central point of this challenge. In the project we accepted this view extended with multi-agent view on the architecture of the IDS and its learning components ([InterRep#1], [InterRep#2], [InterRep#3]). That is way the intrusion detection learning is considered here as data fusion learning problem.

Multi-agent system (MAS) view presents an advantageous paradigm for analysis, design and implementation of complex software systems the IDLS belongs to. It proposes powerful metaphors for information system conceptualization, a range of new architectures, techniques and technologies specifically destined for large scale distributed intelligent systems ([Weiss *et al*-99], [Wooldridge-01]). It is necessary to distinguish two aspects of using multi-agent paradigm in intrusion detection learning scope: (1) Use of multi-agent architecture in intrusion detection learning systems (*Multi-agent IDLS*); (2) Use of multi-agent architecture for intrusion, i.e. for distributed decision making (*Multi-agent IDS*). IDLS and IDS definitely fall into the class of potential MAS applications. The main reason is that in these applications data sources are spatially distributed and data processing is performed in distributed manner.

The approach that is being developed in the reporting Project is in some respects close to the approach that is being developed by the group of S.Stolfo. We borrowed from it the idea of meta-classification and idea of using multi-agent architecture for intrusion detection learning system. However, this similarity concerns only with the ideas in general.

The *distinctions* are manifold:

- The *first* of them is in architecture of agents and multi-agent system as a whole. The particular agents are proactive and of different specialization.
- The *second* distinction is that agents interact on the basis of shared knowledge represented in ontology, which includes structured problem and subject domain components. The latter would make it possible to deal with detection not only simple attacks against particular computer, but also with detection of distributed attacks against computer network as a whole.
- The *third* distinction is in techniques for knowledge discovery from audit data that we implemented. Together with the standard techniques for learning base classifiers and meta-classifiers we implemented the original ideas and methods including original algorithm of *temporal data mining*.
- Possibly, *the most important distinction* of the approach accepted in this project (as compared with the approach used by S.Stolfo's group) is that we consider network intrusion detection task as *multi-sensor data fusion*. This view was firstly discussed in [Bass-00]. Actually, to provide reliability and required quality of IDS operation and its cooperation with network management system, network-based IDS has to collect and to process information from numerous heterogeneous distributed sensors monitoring input traffic, audit trails, operational system, servers, directories, databases of user profiles, etc. These sensors should be situated in different hosts of the network and measure and compute numerous characteristics on the basis of analysis of the network traffic and audit data trails. These characteristics are very diverse from several points of view. They can present *IP* packets and their components, symbolic data measured in categorical scale, temporary ordered sequences and subsequences of events with attributes, real-valued data. This data can be of different generalization level. Some data are derivative due to preprocessing and high-level computations. In order to detect attacks against a particular host or against the computer network as a whole, it is necessary to solve large-scale data fusion task [Bass-00].

The following material of the Report is organized as follows.

The rest of the *Chapter 1* is practically devoted to the methodology of distributed intrusion detection on the basis of heterogeneous distributed data sources. Firstly it reviews the structures of the available learning data and its use in attack detection and learning and afterwards "builds a bridge"

from peculiarities of available data to methodology of distributed intrusion detection and intrusion detection learning. . In general, the objective of this Chapter is to make it clear the peculiarities of data to be mined and specificity of the intrusion detection learning based on multi-agent architecture. Particularly, it discusses the existing principles of data and information fusion and motivates the choices adopted, analyzes the potential structures of distributed decision making and decision combining, considers the developed structure of IDLS distributed knowledge base and the training and testing methodology with accent on peculiarities entailed by distributed nature of training and testing and also describes a methodology of allocation and management of training and testing datasets.

Chapter 2 describes the technology of multi-agent IDLS design, implementation and deployment. Two software tools intended for support of the engineering processes are presented. The first of them called *Multi-Agent System Development Kit* (MASDK) supports design, implementation and deployment of the reusable components of MAS while the second one, *Information Fusion Design Toolkit*, mainly supports design procedures oriented to design of distributed data and information fusion–oriented functionalities. The Chapter also describes the developed ontology specifying high-level representation of the basic notions of Intrusion Detection Learning domain.

Charter 3 is devoted to the architectural issues of IDLS system from design and implementation viewpoints. The Chapter describes generic tasks resulting from decomposition of the entire intrusion detection learning task and allocation subtasks over particular agent classes. It outlines also generic architecture of agents and points out some peculiarities of the multi-agent technology that was used in design and software implementation of the Intrusion Detection Learning System components.

Charter 4 considers a case study used for prototyping IDLS according to the developed architecture and model by use of the proposed technology. Categories and instances of attacks to be used in case study are described. The proposed "strategy" of multiple classifications implementing detection of intrusions in terms of *multi-step anomaly detection* is described. The data sources and generic data structures used by IDLS components are considered. Examples of training and testing instances are given.

Charter 5 presents the software tools destined for the design, implementation and deployment of IDLS as well as evaluation of these tools.

Concluding remarks provide generalized view on the results of the Project.

Appendixes demonstrate the developed multi-agent technology destined for distributed intrusion detection learning and decision making. For this purposes, while solving an intrusion detection learning task used the developed case study (see Chapter 4), it comments the designers' activity and describes the distributed learning process itself and also intermediate and final results of data mining and knowledge discovery aiming at engineering and validation of the distributed knowledge base of a multi-agent intrusion detection system.

1.2. Main Concepts of Logging and Auditing of Events in Computer Networks. Representation of Audit Data at Various Generalization Levels

Let us consider at first main concepts of logging and auditing of the events in computer networks (event, logging, auditing, audit database, audit record, audit data analysis etc.).

Event is an occurrence formed on the basis of processing of the input message traffic. An event is specified as triple comprising a *subject* (an active object, fulfilling action, for example, user or program), an *object* (a passive object subjected to an action, for example, disk, directory, file, etc.) and an *action* (for example, reading, recording, execution, etc.). Sometimes event can be assigned additional attributes. The following *list of the inspected events* can be used: logon (logoff) of an access subject into (from) a system; generation of printed (graphical) output document; start-up (completion) of a program and process (jobs, tasks); access of an access subject program to defended files, including their creation and deletion; message transmission to data link; access of an access subject program to a terminal, host, data link, peripheral, program, volume, directory, file, record, fields of record; modification of the access subject authority, defended access object, etc.

Any event at the host or at the network level can be caused by *message transmitting*. A message can be transmitted through a legal or an unauthorized access channel. Any message is assigned a set of

attributes. Using values of these attributes, it is possible to determine with preset probability whether the message is transferred through a legal or through an unauthorized access channel, and also whether it belongs to an unauthorized access script.

Logging represents a process of the security related event sequence registration in the defended network resulting in log data, or *audit data*.

Auditing is a process aiming at analysis of the log data. Analysis aims to detect attacks and unauthorized actions, to disclose security system vulnerabilities, to assess user's operation, etc.

The *audit mechanism* enables logging of security related input events occurred. The log, or the *audit trail*, provides the history of system operation that makes it possible for administrator to review the causes of security violations and to trace it back to the user accountable.

Thus, logging and auditing aim together to disclose the malefactor or user responsible for abnormal or suspicious activity, to detect attacks and unauthorized actions, to reconstruct the event sequence, to store information needed for detection and analysis of other network security problems.

For purposes of the Project, logging and auditing are considered as a way to form data about attacks, normal and suspicious activity to use it as learning (training and testing) data in development and implementation of the security system learning mechanisms.

Audit database is the main source of the learning data. It comprises records of sequences of the time-ordered events occurred. The audit data registers the results of network and host (OS and applications) activity and should be such as to provide sufficient data to restore, to look-up and to analyze a sequence of operations executed by someone within the defended network.

Each record in the audit database can be represented in terms of the event (message) attributes and (or) their values. An event attributes are partitioned into two groups. The attributes of the first group specify the event itself (subject, object, access mode, arrival time), and the attributes of the second group specify results of event (message) processing (processor usage time, size of information input in a data link, etc.).

Normally an *audit record* specification comprises the following positions: record type, date and time of an event occurrence (time stamp), name (address) of the target host, identifier of the user initiated an event, event type, result of the event execution, etc.

Typically an audit record specifies the subject in terms of user name and identifier, group membership identifier, process identifier and terminal identifier. As a rule, the action identifier specifies the event. Name, identifier, access permission and locations typically specify an object.

Most of event-associated actions within computer network are not atomic; instead, they generate multiple secondary events. The latter corresponds to different processing levels however in audit record they can be presented by single event. For example, a single write operation might generate multiple events including “request for a write operation” event, “initiation of a write operation” event, a number of “status of a write operation” events, and “completion of a write operation” event. However often, only a single “write” event is recorded in the audit trail. The type and timing of an event recorded in an action define the types of information that may be extracted from the audit trail with regard to an action. If a forepart of the operation sequence is only recorded in audit trail as the event then the outcome of the operation sequence would not be deducible from it. If an event is generated only at the completion of the operation then suspicious operations can be detected too late.

The logging and auditing subsystem should realize the following procedures: collection and storage of the audit data (network traffic packets, system calls, operating system or application commands), data fragments integration, event extraction, audit record creation and audit data analysis.

Collection and storage of audit data supposes recording the following data items: data to collect and to store, period of cleaning and archiving of audit database, degree of data management centralization, storage location and tool, possibility of ciphered recording of information, etc. The audit data should be protected, first of all, against unauthorized modification and, probably, disclosure. Integration is necessary for unification and coordination of the recorded data formats received from different sources.

One of the most important functions is *audit data analysis*. For detection of attacks and unauthorized access, two classes of methods can be used: (1) statistical and (2) heuristic (or signature-based). *Statistical methods* compute averaged values of parameters representing performance of the computer software components (so-called “historical” profile of traffic) with subsequent on-line

matching them against current values. Remarkable deflections can serve as evidence in favor of attack like directional storm of inquires, fast spread computer virus, implantation into a system of an infringer masked for a legal user, but behaving differently ("masquerade"), etc. *Heuristic methods* use attack detection rules based on known scripts and patterns peculiar to particular attacks, and on parameters of the monitored system that indicate breakdowns or infringer's actions aiming at unauthorized access. Heuristic methods are only able to detect known threats specified in the knowledge base of IDS.

The *main data sources* of audit data are as follows ([Bace-00], [Proctor-01]):

(1) Host-based sources (or system subjects' activities - users, OS, applications, etc.);
(2) Network-based sources;
(3) Additional sources (out-of-band sources, for example, users' messages, telephone switches, physical security systems, etc.).

The first source corresponds to the real time actions performed by subjects of the defended system (users, OS applications, etc.) on hosts. The actions recorded in the audit data have to be selected among those that are the most representative in line with the security policy used.

Network traffic is one of the primary data sources used to detect violations of the security policy. Network traffic consists of the transmitted network packets (frames). A standard network traffic packet consists of three components: packet header (overhead data, sender and destination address, and other fields); packet data fields; packet ending (checksum, delimiter, etc.).

In most cases, *additional data* are derived from users' messages on the basis of analysis of the software (for example, the network service software), analysis of the processes executed, target file types that can provide an evidence about attack (normally, digital "fingerprints" and banners), etc.

It is noteworthy to note that information from a single source cannot often be a solid argument in favor of an attack or a violation of security policy. Results of fusion several data sources (preferably independent from each other) lead to a more certain and reliable conclusion.

Let us describe in more detail the first and the second levels of audit data representation ([InterRep#1], [InterRep#2]).

Host-based audit data sources consist of three main sources ([Bace-00], [Proctor-01], [Amoroso-99], [Northcutt-99]):

(1) *Operating system audit trails* (records of system events generated by particular operating system mechanism, and records of operating system kernel events – system calls);
(2) *System logs* (file of records of system and application events normally registered in text format by special programs);
(3) *Audit data of applications*.

Operating system audit trails are generated by particular auditing software of the operating system. Audit trails are a collection of data reflecting operating system activities ordered chronologically and recorded into *audit file(s)*. Audit file is composed of *audit records.* Each record corresponds to a single system event. The records represent user actions and processes (commands) invoked on behalf of users. The commands can be local or remote. Each audit record is composed of a series of *audit tokens* describing the fields within the record [Bace-00]. Operating system records events at kernel level (reflecting system calls) and at user level (reflecting application events). Audit records contain information concerning the subjects responsible for the event and concerning the objects associated with the event. Most records also include information about the process initiating the event, and the *userID* associated with the event. The latter sometimes includes current *userID* together with the original *userID* (in case of the user identity changes). Kernel-level entries contain system call arguments and return values, whereas user-level entries contain high-level descriptions of the event or application-specific data.

System log is a file that contains records of various system events and settings. UNIX operating systems provide a set of system logs, along with a common service, *syslog*, which supports generating and updating event logs via the *syslogd* daemon. Although a rich lexicon of standard formats and definitions can be used in generating and interpreting *syslog* entries, the security of the logs is considered to be weaker than that of kernel-generated operating system audit trails [Bace-00]. There exist several strategies to consolidate logs to fit it for intrusion detection. The first of them that is still

widely used consists in saving raw logs in a database and in constructing afterwards queries that aim to represent the log data from a variety of perspectives. In some cases, when external evidence indicates that a problem has taken place during a particular time interval in the past, it can be very helpful to simply isolate the logs corresponding to that interval. The respective data can be sorted then according such attributes as user, system object, and occurrence order. In some cases, when analyzing audit trails or network traffic, an administrator or IDS may be able to understand that something suspicious is happened, but not able to confirm or dispel this suspicion. In this situation, synchronizing events reflected in low-level kernel audit records with coarser-grained log events can sometimes help to decide what further exploration and/or responses are required [Bace-00].

Application logs often represent the only available user-level marks of system activity. An example of an application environment in which the need for audit trails and intrusion detection is apparent is the database management system. In many large companies the most critical information resources are housed and accessed strictly via the database management system.

The issue associated with application audit depicts the following intrusion detection challenges:

- *Temporal ordering of audit events*. In IDS, information is normally structured by recording or assigning a time stamp to each event with the following saving the event stream in respective order. If the time stamp has been missed it is hard to discover events that occur in particular time.
- *Composition (fusion) of audit trails from different applications*. It is undoubtedly that this can lead to the strengthening of the intrusion detection mechanisms.

Network traffic is the most common information source in modern IDSs. In network-based approaches, information is collected from the network traffic stream as it travels within the network segment.

In the Internet the TCP/IP protocol stack is used. The TCP/IP stack contains four protocol layers, which are depicted in Fig.1.1.

Each layer adds header information to data of application layer.

The TCP layer adds header information to packets as shown in Fig.1.2.

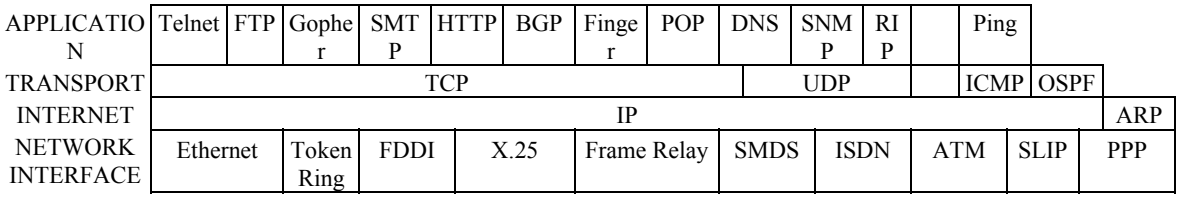


Fig.1.1. Simplified TCP/IP protocol stack

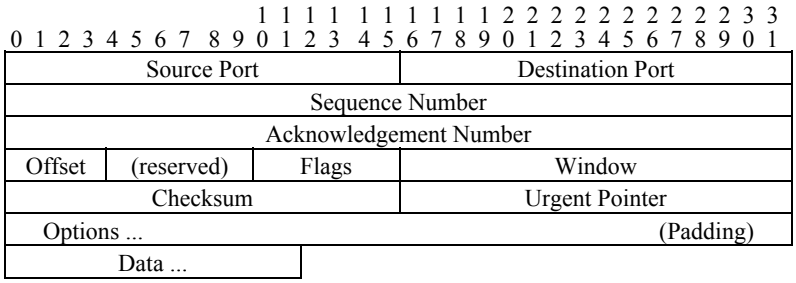


Fig.1.2. TCP segment form

The packet with TCP header information is then forwarded to the Internet layer, where the *IP* datagram header is attached. The content of *IP* packet headers is shown in Fig.1.3.

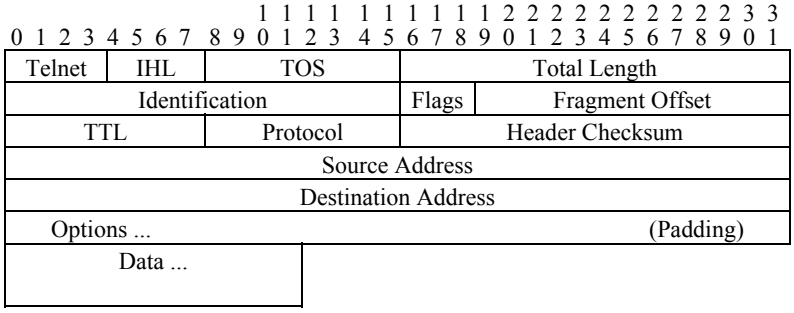


Fig. 1.3. *IP*-packet header format

Finally, the packet is passed to the network interface and sent on to the intended destination, where the *TCP* layer strips off the header information and reconstructs the data stream from the source computer.

The network packets are captured by special software tools called *packet sniffers*. For example, such packet sniffers as *Microsoft Network Monitor* and *Windump* can be used with on Windows platform. There are a lot of different monitors used in Unix, for example, *TCPdump*, and *Arpwatch*. For example, *TCPdump* is a network monitoring and data acquisition tool that performs filter translation, packet acquisition, and packet display. The filter translation function provides a high-level language for specification of filters, along with a user-transparent compiler and optimizer. In addition to packet sniffers, different other network devices can yield information that affects the detection of security problems. For example, a network management system can provide performance and utilization statistics that are extremely helpful for understanding whether a detected problem is likely to be security-related or due to other system factors.

One of the main *additional data sources* is users' messages recorded manually. In some cases they will help discover problems that are not detectable by other means. Another additional source is the network service *banners*. Many network services in response to any query send definite reply called a header, or a banner. The analysis of the banner data (e.g., the version number) provides opportunity for conclusions about whether or not the network service is vulnerable. *Digital fingerprint* is based on the comparison of the fingerprint of a fragment of software to a fingerprint of a known vulnerability. The antivirus software that matches fragments of the scanned software against virus signatures using virus detection database also uses this process. The varieties of this method are checksums and time stamps of the analyzed software. A discrepancy between the current and the standard values indicates the changes made to the checked object (program or data file) that could be result of an attack.

Examples of the audit data corresponding to different data sources were considered in [InterRep#1] and [InterRep#2].

1.3. The IDLS Data Sources Taxonomies

The data, which can be used for intrusion detection learning, exist in many different forms and can be received from different sources.

This information are gathered immediately from network traffic, from logs of operating systems, logs of DBMS, logs of different applications, alerts and system messages of monitoring subsystems, programs analyzing executable processes for unexpected behavior, programs detecting changes of directories and files, etc.

For systematization of data sources, which can be used for intrusion detection learning, we shall consider their taxonomies.

The taxonomy of data sources of possible intrusions into computer networks is a classification scheme, which structures knowledge of this subject domain and determines the relations between units of knowledge.

Let us select three main tags (characteristics) for distinguishing data sources:

(1) location of source and software generating data;

(2) level of data processing;

(3) an object, with which the data are associated. The data sources *taxonomy,* which classifies data sources *due to location of source and software generating data*, is as follows:

(1) Network-based data sources;
(2) Host-based data sources;
(3) Other data sources.

Let us differentiate *network-based sources* depending on network layers and used protocol. So the following sources can be distinguished:

- Link layer (Ethernet, Token Ring, FDDI, X.25, ATM, Frame Relay, ISDN, SLIP, PPP);
- Network layer (IP, ARP);
- Transport layer (TCP, UDP, ICMP, OSPF);
- Application layer (FTP, TELNET, Gopher, SMTP, HTTP, SNMP, BGP, *r*-service, *X*-Windows, DNS, RIP, etc.).

Host-based sources of data are as follows:

- *Operating system (OS) audit trail* (system event records sequenced in chronological order and generated by an OS auditing software);
- *System logs* (usually text files representing system and application events written by system programs as a record/per time discrete). Examples of such system logs are log of commands running by users indicating the resource used, most recent successful/ unsuccessful login for each user, log of all login failures, all use of *su* command, log of all user logins/logouts and system startups and shutdowns, etc.
- *Application-related audit data* (FTP log, TELNET log, Mail logs, HTTP log, DNS log, Firewall logs, DBMS logs, etc.).

Other data sources can be represented by users' messages informing about problems met; data derived from telephone switches, physical security systems, etc.

The data sources taxonomy, which classifies data sources due to processing level, can be defined by three sources:

(1) Primary (raw) sources;
(2) Preprocessed (filtered, cleaned) sources;
(3) Generalized sources.

Primary (raw) sources are network traffic (packets), host command (system calls) traffic, and data from other sources.

The following preprocessed (filtered, cleared) sources can be selected: *tcpdump* (for packets), preprocessed (cleaned) OS audit trail, system logs, and audit data of different applications.

Generalized sources results from computing by statistical processing programs (SPP). *The data sources taxonomy, which classifies data sources due to an object, with which the data are associated*, is the following:

1. For network-based sources:
   - *Packets;*
   - *Connections;*
   - *All network traffic.*
2. For host-based sources:
   - *Traffic within a connection;*
   - *Processes* (Logins/logouts, System startups and shutdowns, Opening/closing of files, etc.);
   - *Users* (Remote users, Local users);
   - *Files and directories* (System files and directories, Users' files and directories);
   - *Disks*; *System registry,* etc.

Let us select data sources that are supposed to use in IDL case study. These are as follows: (1) Network-based sources:

- *Tcpdump* (preprocessed IP, TCP, UDP, ICMP packets);
- *Tcpdump-based* statistical data.

(2) Host-based sources:

- Preprocessed OS audit trail and statistical OS audit data;
- System logs (for example, log and statistical data of commands run by users plus resource; Log and statistical data of all login failures; log and statistical data of all user logins/logouts and system startups and shutdowns);
- Application audit data (for example, FTP logs and FTP statistical data, TELNET logs and TELNET statistical data, mail logs and Mail statistical data, HTTP logs and HTTP statistical data, DNS logs and DNS statistical data).

1.4. Features of Audit Data used for Knowledge-based Attack Detection

In order to be able to detect attacks in progress of accumulation of the audit data it is necessary to identify them and differentiate from regular events. The list of main common patterns for knowledge-based attack detection can be as follows ([Amoroso-99], [Lukazky-01]):

(1) Repetition of a suspicious action;
(2) Mistyped commands or responses during an automated sequence;
(3) Searching for and exploitation of known vulnerabilities;
(4) Inconsistencies in traffic parameters and contents;
(5) Unexpected attributes of some service request or packet;
(6) Unexplained problems in some service request, system, or environment.

Any means of protection (firewalls, authentication servers, access control systems, etc.) use one or two of the conditions listed above, while IDS (depending on the implementation) must use all of them.

Repetition of a suspicious action. One of the best ways to detect an attack is to detect a *repetition of a suspicious action*. This mechanism is based on a premise that if a malicious party does not know how to get access to a resource at first attempt, it will try again. Examples of such actions include scanning ports in search for available network services or fitting passwords. The illegal activity detection algorithms should be able to detect such repetitions and decide after how many additional attempts the conclusion can be made about the attack. It should be noted that if a malefactor knows how to access a resource from the very beginning (or is able to intercept or fit an identifier and password of an authorized user) and makes no mistakes, then this illegal intrusion will be virtually impossible to detect. If the malefactor is able to create a testbed simulating the system he will attack later, and practices on this testbed, aiming to imitate the patterns of authorized users, then it is often impossible to detect such illegal activity. Detection of repetitive actions is a powerful approach, because it helps detect unknown attacks of unknown types.

There are three methods of repetition detection:

- *Thresholds control.* In this case a certain threshold is controlled that allows distinguishing between authorized and unauthorized repeats. Unauthorized repeats may correspond to both regular errors and real attacks. In any case, all instances going over the threshold will be detected. The practical operational experience allows for finer tuning of threshold values for different system parts as opposed to default threshold values. A typical example of threshold values specifying is the number of password input attempts. A wrong choice of a threshold value may lead to either the false negative or false positive problem. In other words, a threshold value that is too small will lead to triggering the attack detection system too often (false detection), and a threshold value too great may lead to leaving some of the attacks undetected.
- *Control of the time between repeat instances.* A typical example is detecting port scanning, i.e. a given number of a node's port accesses per specified time interval. It is noteworthy that a wrong choice of the time interval may also lead to the false negative and false positive problems.
- *Repetitive patterns control.* Let us look at an example of this method. Through the *SYN Flood* attack, a well-known hacker Kevin Mitnick was able to disrupt the operation of the Tsutomu Shimomura's computer ([Northcutt-99]). In this example, a connection request (sending of a *SYN*-packet) is viewed as pattern. Repetitive connection requests lead to the host queue

overflow, and the node being unable to receive new requests. In this example, the overflow took place after 8 requests for login service (port #513).

Mistyped commands or responses during an automated sequence. Another method of unauthorized activity detection consists in detecting *mistyped commands or responses during an automated sequence*. Inconsistency between expected and actual responses leads to the conclusion that one of the parties in the information exchange has been substituted: either the requesting party, or the responding party. For example, information about the *sendmail* system handshake procedure between the mail processes is kept in the Unix event log. Command sequence in this procedure is predictable. Audit data can reflect the fact that one of the processes sent mistyped requests or responses. This may be caused by the malefactor's attempt to replace the mailing system software. A classic example of such attack is shown in [Amoroso-99]. The malefactor used a widely known vulnerability in the debugger mode of the *sendmail* program. He then tried to obtain password file for the mail gateway, but made a typing mistake (symbols ^H), which caused detection.

Searching for and exploitation of known vulnerabilities. An attack detection system should detect *searching for and exploitation of known vulnerabilities*. The use of automated search tools for widely known vulnerabilities (the so-called security scanners) typically constitutes a separate category of attack characteristics. There are a number of such tools, including both freeware utilities (*Nmap*, *Queso*, or *SATAN*) and commercial software (*Internet Scanner*, *Cisco Secure Scanner*, etc.). The fact that a security scanner is being used does not necessarily mean that an attack is in progress. These tools may be used for scheduled security system component checks. Therefore, an additional analysis of all registered occurrences of security scanner usage must take place. For example, if a short time after a scheduled system scan another vulnerability scan is taking place on the same system, this may mean that an attack is being undertaken.

Inconsistencies in traffic parameters and contents. Certain *inconsistencies in traffic parameters and contents of network traffic* may also serve as indication of attacks.

Let us consider main types of these inconsistencies:

- *Input external packets with internal source IP addresses.* In case an attack detection system or another access separation tool (firewall or router) cannot control the traffic direction, then an "*address spoofing*" attack may be undertaken. This attack allows the malefactor to perform illegal actions if they were coming from one of the nodes internal to the system, which usually have less strict security requirements.
- *Output internal packets with external source IP addresses.* In this case, a malefactor would try to make his activities in a mode to look as if activities were performed by a user external to network, to send any investigation on the false track and to avert any suspicions from the internal users.
- *Unforeseen packet addresses*. In this case, packets with unforeseen sender address (or receiver port for protocols based on TCP/UDP) may serve as attack indications. The first example of this case is a detection of the input packet with the *IP*-address that is inaccessible or impossible for the external network. E.g., there are addressees that are not routed in the Internet. Among such addresses are: 10.*.*.*, 172.16.0.0 — 172.31.255.255 and 192.168.*.*. Their description is found in RFC 1918 ([RFC1918-96]). Besides the addresses listed above, there are a number of address ranges, from which no packets can come into your network. The second example is a *Land* attack, in which the sender port and address are the same as the receiver port and address. Processing such a packet leads to infinite looping. The third example can be a connection request by Telnet protocol from an unknown node or from a node with which there is no trusting relationship.
- *Unforeseen parameters of network packets.* There are many attacks with *unforeseen parameters of network packets*. The number of such attacks keeps growing, as malefactors keep finding vulnerabilities in the TCP/IP stack implementations in different OS. Any packet that does not comply with the RFC standard may lead to the malfunction in the communication equipment that processes that packet, including not only routers or switches, but also firewalls and IDS. Many attacks use illegal combinations of *TCP*-flags in network packets. Some

combinations lead to malfunction of the host that processes such packets; other combinations leave such packets undetected by some attack detection systems or firewalls. [RFC0793-81] describes how different systems should respond to normal *TCP*-packets. However, RFC documents do not say how a system should respond to incorrect *TCP*-packets. As a result, different devices and OS respond differently to *TCP*-packets with illegal combinations of *TCP*-flags. Illegal combinations can be detected by at least one of the characteristics listed below [Frederick-00]:

- *SYN* + *FIN*, for these two flags cancel one another out. The first one establishes a connection, while the second one ends it. Now many systems register such flag combinations. However, the addition of another flag to this combination (e.g., *SYN* + *FIN* + *PSH, SYN* + *FIN* + *RST, SYN* + *FIN* + *RST* + *PSH*) leads to some attack detection systems not being able to detect this altered scanning.
- *TCP*-packets should never contain just one *FIN* flag. Typically, a single *FIN* flag is an indicator of *stealth FIN*-scanning.
- *TCP*-packets should have at least one flag (but not *FIN*).
- If a *TCP*-packet does not have the *ACK* flag and that packet is not the first one in the three-way handshake then such packet is definitely incorrect, i.e. any *TCP*-packet should have the *ACK* flag.
- Other suspicious combinations are *RST*+*FIN*, *SYN*+*RST*.

In some protocols, for example, in TCP, there are bits that are reserved for the future expansion of the protocol. Presently these bits are not used, and therefore an appearance of network packets in which they are used may mean unauthorized activities.

Other indicators of an incorrect network packet include:

- Sender or receiver port is 0 (for *TCP*- and *UDP*-packets);
- For *TCP*-packets with the *ACK* flag, the acknowledgement number can never be *0*.

Very often, the network packet size that is larger than normal can be an indicator of an attack. For example, most of the ICMP Echo Requests have an 8-byte header and a 56-byte data field. When packets of irregular length appear, this may be an indicator of illegal activity. E.g., the *Loki* attack allows tunneling different commands into the ICMP Echo Requests and responses to them into the ICMP Echo Replies, which changes the data field size substantially compared to normal. Another example could be the *Ping of Death* attack, which generates an ICMP Echo Request with the packet size over 65,535 bytes. This attack is particularly dangerous together with fragmentation of an ICMP request.

Another classic indicator that allows detecting an attack in most cases is the appearance of fragmented network packets. Many network security tools are not able to collect fragmented packets correctly, which leads either to the malfunction of these tools or to such packets getting into the network under protection. The former is achieved through fragments with incorrect displacement, and the latter – through the so-called *tiny fragment* attack. This attack creates two *TCP*-fragments. The first one is so small that it does not even include a full *TCP*-header, and more importantly, it contains no receiver port. The other fragment contains the rest of the header. Many firewalls allow the first or both of the fragments to enter into a corporate network.

- *Network traffic anomalies*. Network traffic anomalies are any deviations of the network parameters from the predefined standards. Among them, the following parameters can be taken into account: (1) load factor, (2) typical packet size, (3) average amount of fragmented packets, etc. Any discrepancy may be characterized as an attack, e.g., a *denial of service*, or as *regular network problems* caused by network equipment failures.
- *Suspicious characteristics of the network traffic*. The following are some of the suspicious characteristics that may be indicators of attacks:
  - *Suspicious traffic from/to a specific address*. In some cases, certain types of traffic or its contents may be suspicious. It may be email messages containing the keywords “job search”, access to servers www.job.ru, www.recruitment.com, www.vacation.com, or running a protocol that is not expected to be used by the certain address (e.g., Telnet requests from a bank operations clerk).

- *Suspicious traffic regardless of address.* Certain types of traffic are suspicious regardless of address, e.g., appearance of unregistered protocols in the network, or internal traffic coming from addresses that don't belong to the network. In a Moscow bank, the *RealSecure* system was able to detect unauthorized use of modem by one of the employees who accessed his office computer from home and thus gained a higher-speed Internet access [Lukazky-01]. Another example could be a transfer of highly confidential corporate materials outside the corporate network, which can be detected by monitoring the contents for the certain keywords (e.g., 'Confidential', 'Top Secret', etc.).

Unexpected attributes of some service request or packet. Requests by any system, network, or user are characterized by certain attributes that describe the system, network or user profile. Such profiles are used for monitoring and analyzing the object under control.

The most frequently used parameters that are helpful in identifying potential attacks are as follows:

(1) *Time and date*. Time and date are attributes often used to detect violations of security policies. If, for example, the attack detection system registers an access to the system after the specified time, on a weekend or during holiday time by a financial department employee, this could serve as a reason for additional investigation. The employee is not performing well enough, or they are working on a quarterly report, or somebody is trying to undertake an attack using their login. There is also another example of using the time as attack indicator: if the time period between the transaction request and confirmation is too short to properly acknowledge the correctness of the transaction, this may be an indication of a fraud or an incorrect transaction.

(2) *Location*. Usually a user logs into the system from the same computer, or accesses the Internet through dial-up connection from the same telephone number. Thus, logging in from a different computer or telephone number may be viewed as an anomaly. In a large network that is spread spatially, the physical location at the time of access may also be monitored. Thus, a non-typical location of access may be an indication of an attack.

(3) *System resources*. Characteristics of many system resources can also be indicators of attacks. For example, an intensive higher-than-average load of the CPU may mean that various illegal activities are taking place. This could occur either because of irregular operation of the system or of some application, or because of an attack (e.g., the *brute force* attack). Other characteristics of resources used to detect attacks include intensified use of RAM/HDD memory, files, ports, etc.

(4) *Service requests*. Another method of detecting attacks consists in the analysis of services that are most often requested by the subject (user, process, etc.) in their regular activity. E.g., if an employee needs Internet access to do their job, then a list of most often needed Web servers can be made, and then connections to servers that are not on the list may be viewed as violations of security policies. Requesting certain files, sending and receiving of e-mail messages to/from certain addresses, using certain services (e.g., FTP or Telnet) and other types of activity may also be indicators of security policy violations.

(5) *User and system profiles*. The usage of profiles is a more general approach than the analysis of services or system resources. User and system profiles include requests for these services and resources. However, they also have new parameters that are specific to the user, process, or node (e.g., peak and lowest load times, typical session duration, typical login and logout times, etc). Discrepancies between the current and the typical values of such parameters pertaining to the system under protection could indicate anomalous behavior.

Unexplained Problems in Service Requests, System, or Environment. Any problem in the protected network should be a reason for additional investigation. Some of such unexplained problems are as follows.

- *Software and hardware problems.* Router failure, server overload, or impossibility of starting of a system process/service may be an indication of a "*Denial of Service*" attack.
- *System resources problems.* A sudden shortage of disk space may indicate a "bomb" in he system (a small archived file that takes up hundreds of megabytes after being extracted).

- *Processing power problems.* Delays in Internet gateway or application server responses may camouflage the "*Denial of Service*" attack.
- *Unexplained user behavior.* Unexpected request for a resource that has never been requested before may indicate the fact that a malefactor has intercepted or fitted a password of an authorized user and is now trying to access some important information.

1.5. Basic Data Structures and Measurement Scales used for Data Representation. Dimensionality and Size of the IDL Training and Testing Data

The data structures are characterized by relationships met by data items. They concern, in essence, to "space" concepts: they can be represented to the scheme of organization of data in computer memory. The important tag of data structure is a character of order of its units.

In practice the following *typical structures of data* are used for IDL:

- *Time-based sequential* (temporal, time-stamped) data. The specific feature of most audit data (independently from level of its specification) is that each event of audit data flow is time stamped. Time stamps can be measured in terms of real time or in terms of a discrete time sequence);
- *Sequential* (ordered) data (when data are ordered, but time stamps are absent or non-relevant);
- *Relational* (non-sequential) data;
- *Transactional* data. These data describe different elements of some process, for instance, commands of operating systems fulfilled as a single unit (all or anything) and transforming system from one state to another.

The usual is to represent data utilized for training a classifier in the form of a table. In such a table each column is mapped by the name of an attribute and each row corresponds to an "object" ("instance", "case"), for example, network packet, OS command, connection, etc. The tuple of attributes of each table row corresponds to the particular object and is mapped to the "interpretation" that is the label of a category (class) to which the object belongs. For example, interpretation can be the status of connection (normal, abnormal), name of attack, etc. It is admissible that each table row can be mapped to several categories. In the last case each of the categories corresponds to a particular classification. An example is the case of hierarchical classification, in which each category corresponds to the classification at a particular level of generalization-specialization. Data structured according to a classification decision tree is one more example of the case, in which every line of the respective data table storing data is assigned several interpretations.

The most significant property of each attribute is its *measurement scale* or "*data type*". There exist several measurement scales of data ([Suppes *et al*-63], [Pfanzagl-71]).

The *typical measurement scales* that are used for representation of *IDL* data are *Binary* (or Boolean), *Categorical*, *Linear ordered* and *Numerical*. Analysis of measurement scales and their properties was done in [InterRep#1].Thus, the task of intrusion detection learning to create distributed knowledge base content concerns extraction of knowledge from large or very volumes of distributed heterogeneous data and this circumstance in the main one determining the peculiarities of the developed engineering methodology described below.

1.6. Design Principles and Methodology used in IDLS and IDS

Intrusion detection (ID) objective is to detect illegitimate operations of computer users or/and inside and outside attacks on the basis of data and information received from multiple sources of the computer network. Although the ID task is understood as *classification* of status of use of the computer network, it possesses some distinctive peculiarities as compared with the conventional classification tasks. The first distinction lies in the fact that ID in its nature is *distributed classification* task. The second considerable difference is determined by the *heterogeneity* of the data represented in the different sources, and by their possible incompleteness. The third important distinction is that different attacks reveal themselves in different subsets of data sources, have different "life time" what considerably affect on the technology of intrusion detection-associated classification.

Intrusion detection learning (IDL) is a task of distributed data processing aiming at creation of distributed knowledge bases and distributed decision making mechanisms of intrusion detection systems. Within the Project, it is assumed that knowledge needed for ID is formed through using the data mining and knowledge discovery technology, based on the accumulated experience represented in the form of interpreted dataset used for training and testing of IDS components. Technology destined for engineering of distributed knowledge bases through coordinated use of data mining and knowledge discovery techniques is one of the central tasks of the Project.

An essential *peculiarity* of the intrusion detection-oriented data mining and knowledge discovery technology is that in this technology it is necessary to take into account many heterogeneous sources of data and information. Each such a source can only partially specify user activity, both normal and abnormal. This is why the detection of abnormalities in usage of a computer and intrusions into computer network can be successful if only the detection technology is capable to fuse information from many heterogeneous sources and combine decisions reflecting the "traces" of abnormalities "to restore" the complete "picture".

While summarizing *the properties of data* available for IDL and ID (see sections 1.2–1.5) and also *peculiarities of the ID task* itself, the following aspects of the above data and task entail the most considerable difficulties in IDLS and IDS engineering:

- *Formidable diversity of attacks*: a great deal of existing attack types, potential diversity of ways of realization of each of them and also increasing number of new attacks daily invented by malefactors.
- *Multiplicity and diversity of data sources reflecting users' activity*: information can be got from numerous heterogeneous sensors monitoring input traffic, audit trails, operating system, servers, applications, directories, databases of user profiles, etc. (see Fig.1.4).

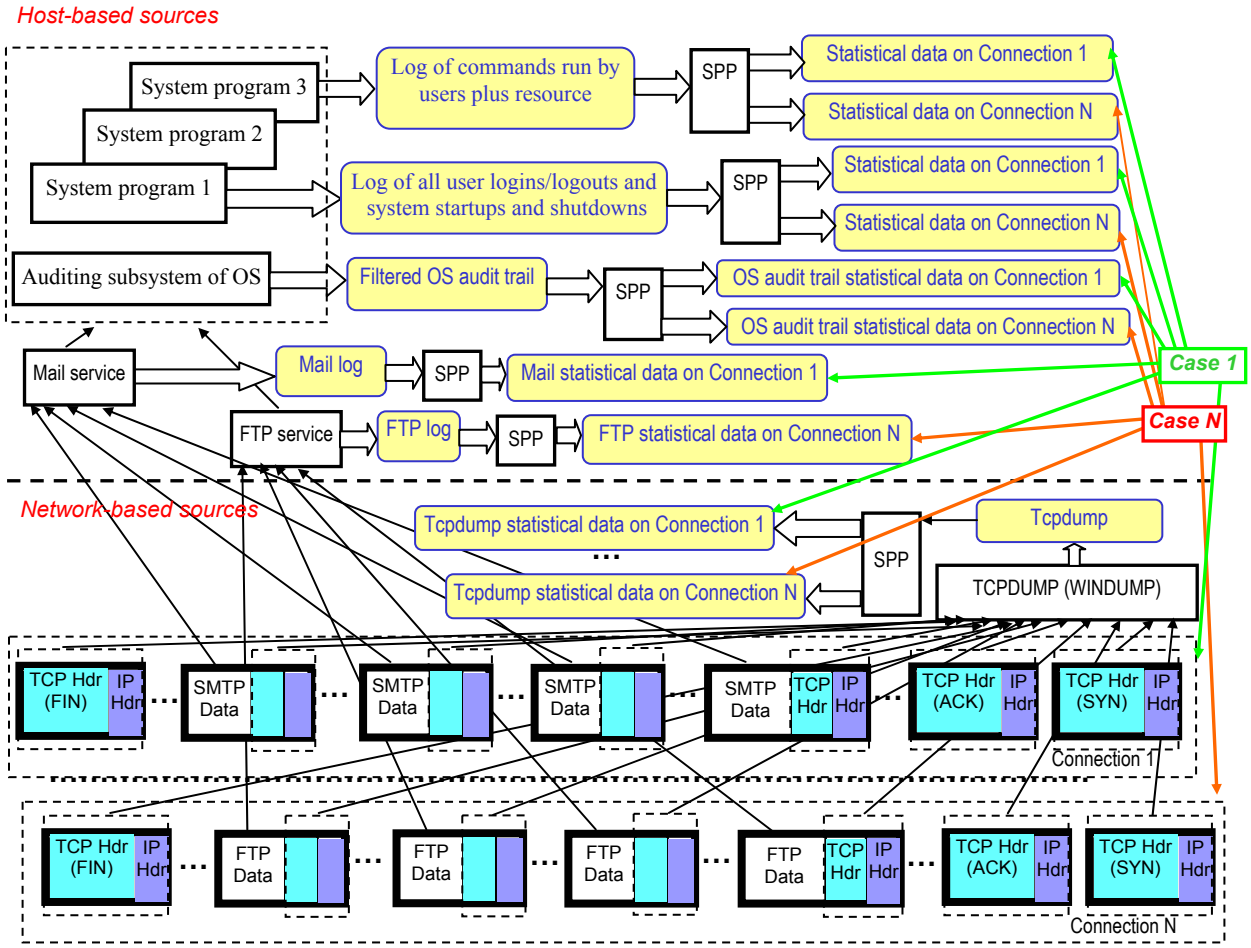


Fig.1.4. Illustration of the diversity of data sources and essence of the data identification problem

- *Large size and dimensionality of learning data*: sensors measure and/or compute numerous characteristics in real-time mode with high frequency.
- *Diversity of data sources from several viewpoints*: *IP*-packets and their components; symbolic data measured or computed in categorical, Boolean and numerical scales; temporal sequences of events with many attributes; data represented at different levels of abstraction (generalization); data derived from raw data via preprocessing and high-level computations, etc. (see Fig.1.4).

The above peculiarities of the task itself and learning data entail several new *difficult problems* which at glance seem to be aside ones for IDL and ID technology but in practice they considerably influence on both technologies [Goodman *et al*-97]. Let us briefly consider these problems.

The *first problem* associated with any IDL system engineering is development of the *shared thesaurus* providing distributed software entities of the system in question with *monosemantic understanding of the terminology* used for formal specification of intrusion detection domain. This problem arises due to the fact that specifications of particular data sources can being developed by particular experts. In most cases these processes are executed in parallel and independent mode. Therefore, experts can denote different domain entities via the same terms and vice versa, they can denote the same entities by different terms.

The *second problem* arises due to the fact that in different sources the same entities can be represented in different data structures, but in meta-level all of them can be understood and used equally.

According to the modern understanding, to cope with the both above problems it is necessary to use *meta-level specifications* of data and knowledge shared by all software entities of the ID system. As a rule, such meta-level specifications are built in terms of ontology comprising *problem ontology*, *application ontology* and *task ontology*. In the Project exactly such an approach is used. However, creation and maintenance of consistent ontology in context of the above peculiarities requires development of a new functionalities and special software. In the developed architecture this task concerns the area of responsibility of several agents dealing with local and global components of the application ontology.

The *third problem* corresponds to a so-called *entity identification problem* [Goodman *et al*-97]. Each local data source specifies an entity (object to be classified) only partially. Its complete specification is made up of data fragments distributed over the data sources. Therefore, a formal technique to identify such fragments is needed to make possible retrieving, collecting and analyzing together distributed data about the same user activity. It is noteworthy to notice that some fragments of data associated with the above entity can be absent in some sources.

An explanation of this problem is given in Fig.1.4 and also in Fig.1.5.

The above problems together constitute the so-called *data non-congruency problem* [Goodman *et al*-97], which considerably influences on the conceptual model, algorithmic basis and architecture of both ID and IDL systems and on the methodology of their engineering.

1.7. Methodology of Multi-Agent Intrusion Detection Learning

Information fusion (IF) methodology developed for IDL systems is constituted by *basic conceptual solutions* with regard to the following engineering aspects:

(1) *Basic principle of data and information fusion*, that is how to allocate data and information processing functions to data source-based level and meta–level executing some processing and decision making functions in centralized mode.

(2) *Decision fusion (DF) meta-model* that determines structure of decision making and combining in IDS and IDLS.

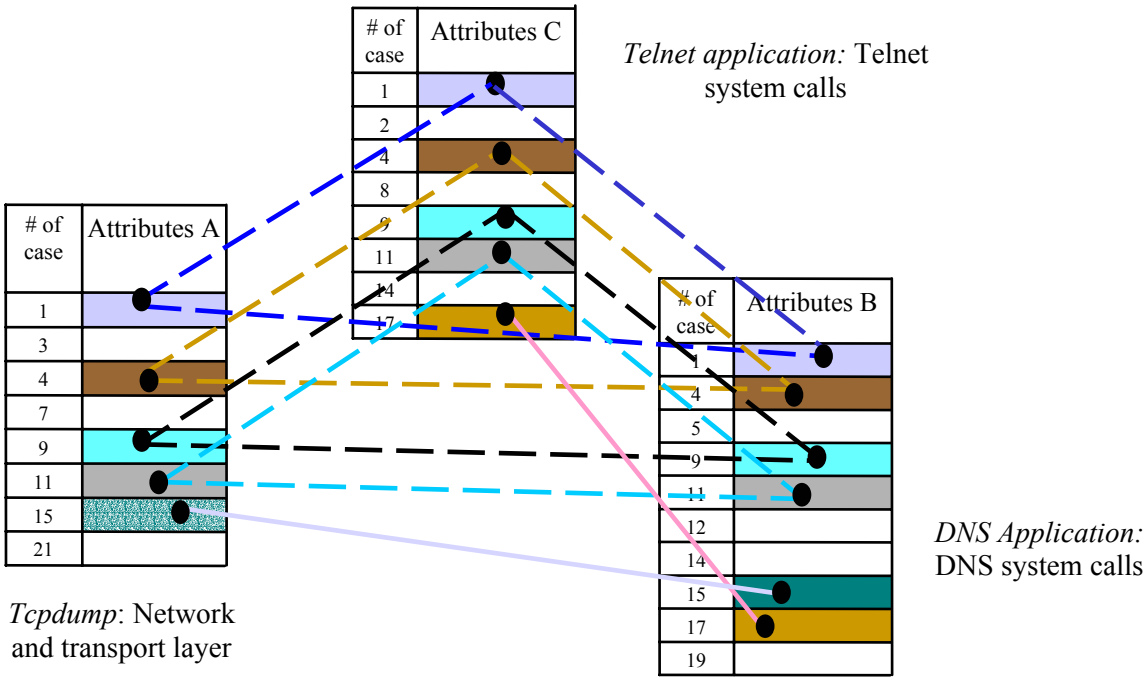


Fig.1.5. Illustration of the essence of the data identification problem

(3) *Structure* of IDS *distributed knowledge base* (KB), its interconnection with *ontology* and structure of *decision making mechanisms*.

(4) *Particular data mining and knowledge discovery techniques* used by IDLS for engineering of distributed KB and decision making mechanisms of IDS.

(5) Particular *techniques* used for *decision combining*.

(6) *Methodology of Training and Testing* Management.

All these aspects of the methodology were described in different details in [InterRep#2]. Let us summarize this description in this section.

1.7.1. Basic Principles of Data and Information Fusion

There exist several variants of *allocation of functions to source-based and centralized levels* of data and information processing proposed for IF ([Goodman *et al*-97]):

- *Centralized data and information processing.* This variant supposes straightforward transmission of data from data sources into a central database for subsequent centralized data and information fusion what on default means that both classification and learning of classification are also performed in the centralized mode. This approach possesses the very obvious drawbacks: (1) inefficiency in the case if the dimensionality of the entire data representation space is too large; (2) very high communication overhead and data duplication. If various data structures are used in different sources then this methodology becomes practically infeasible. In fact, such an approach ignores the very idea of distributed processing.
- *Combining knowledge bases of data sources.* In this case knowledge bases designed on the basis of particular data sources are simply combined in single KB which afterwards is used as KB of a centralized classification system. This model is applicable in case if all local knowledge bases are represented in a common structure, e.g. all of them are rule-based, and the attributes of different sources are in some sense similar. Unfortunately, such an approach is not applicable if data of sources are of large dimensionality and differs very much.
- *Source data based decision fusion.* In this model the decisions produced by local classification mechanisms (base classifiers [Prodromidis *et al*-99b]) are combined in the meta–level. This model is advantageous in many cases, in particular, if there are many data sources and there are representative interpreted datasets sufficient for training and testing of both base and meta–level classifiers combining decisions of the formers. The advantages of this IF model are

as follows: it provides considerable decrease of communication overhead; it is applicable in the case if data structures used in particular sources are very different; there exist a number of effective and efficient methods of combining such decisions in upper level to obtain the final one; it preserves the source data privacy if necessary. This methodology outperforms all aforementioned approaches in many respects. It is accepted in this research as a component of the methodology of data and information fusion.

1.7.2. Decision Fusion Meta-model

In the developed methodology *Decision fusion (DF) meta-model* is composed of three types of structures:

a. *Source-based decision making model.* In the simple case if the dimensionality of the vector of data source attributes is pretty small (about 20–25) and attribute representation structures are more or less homogeneous of source, e.g. are measured either only in numerical or only in discrete scales, then it can by satisfactory to use the single source-based base classifier, whose decision is forwarded to meta–level. In amore complicated case if the dimensionality of the attribute vector is high enough and/or the data of sources are too heterogeneous (measured in different scales, are of different accuracies and reliabilities, have missed values of attributes, etc.), then it is reasonable for such a data source to provide with several base classifiers, such that they produce classifications on the basis of different sets of attributes of trained on the basis of different subsets of training and testing datasets. The decisions produced by these classifiers can be forwarded to the meta–level for combining with decision produced by base classifiers of other sources. An alternative is to combine the data source-based decisions within particular data source (Fig.1.6).

b. *Meta-model of decision combining* ("decision making tree") is constituted by the set of base classifiers of data sources, one or several meta–classifiers and structure given over the base classifiers. In the meta–level, the system combines either decisions of particular base classifiers, or combined decisions of base classifiers, or combination of both above variants. It is important to note that all decisions to be combined are measured either in binary or in cardinal scales. Within classification model used in this research, any classification task for multiple classes is reduced to a number of binary (pair wise) classification tasks and that is why below considered input data of a decision combiner is a binary vector of attributes.

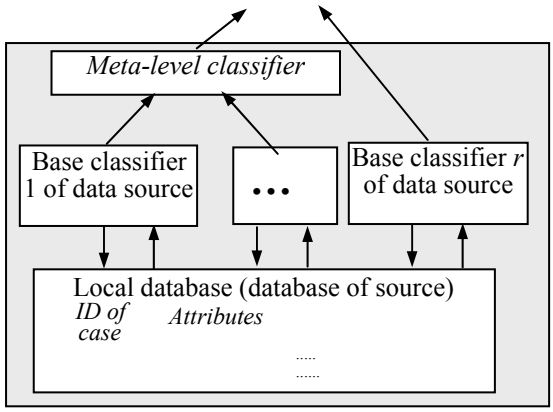


Fig.1.6. Data source decision making model: General case

c. *Meta-model of classification* ("*classification tree*") is used to reduce a classification task in which the number of classes is more than two, to a number of binary ("*pair wise*") classification tasks. To each node of classification tree a decision making task is mapped that includes all the tasks of the base classifiers and also meta–classifiers supposed by decision making tree. Therefore, each node of the classification tree is mapped a decision making tree that is composed of meta-model of decision combining and data source decision making models associated with the particular data sources.

Structure of Decision Fusion meta–model, its components and their interactions are explained in Fig.1.7.

1.7.3. Structure of IDS Distributed Knowledge Base

In the developed methodology *knowledge base of IDS* consists of knowledge bases of particular classifiers performing together information fusion for intrusion detection. It turn, the structure of information fusion comprising *classification tree* and *decision making trees* associated with each node

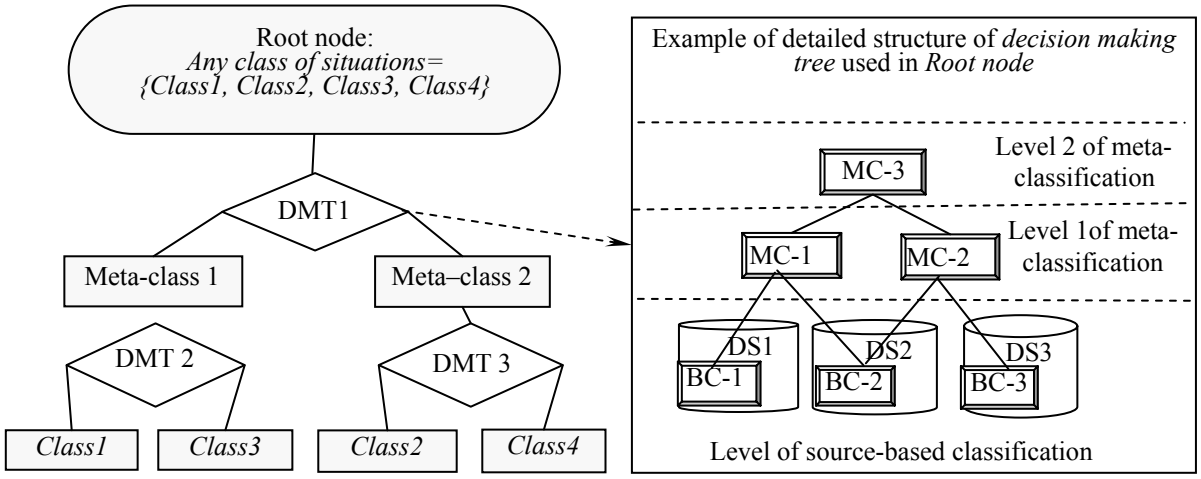


Fig.1.7. Example of Data Fusion meta–model composing classification and decision making trees. Denotations: "*DMT*" – *Decision making tree*; "*DS*"-*Data source*; "*BC*"–*Base classifier*; "*MC*"-*meta–classifier*

of the former unambiguously determines the structure of IDS knowledge base (KB). A KB peculiarity is that it is *distributed* over hosts, in which the data of particular sources are stored and part of KB is situated in a host, in which combining of source-based decisions is performed.

Distribution of KB components as well as their heterogeneity influence on the architecture of IDS and IDLS and also on methodology of their design.

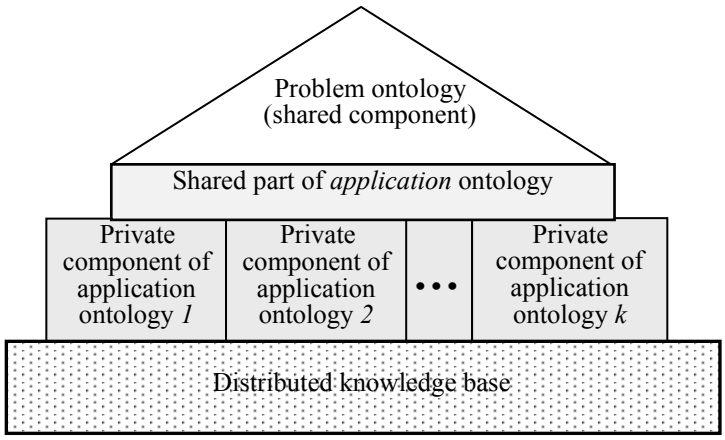


Fig.1.8. Tower of IDLS ontology components

In overall structure of distributed knowledge base the ontology is considered as top-level part of it. Such an understanding of the ontology role makes it easier to effectively resolve the data non-congruency problem discussed in section 1.6. Let us consider the developed structure *of the ontology. Its central component, application ontology*[1] is shared by all software entities of applied IDLS. Shared component of ontology ("shared ontology") explicitly represents all the notions (terminology) used within IDLS and all existing relationships between them thus providing consistent use and interpretations of terminology within IDLS including unambiguous understanding of messages which the agents of multi-agent IDLS exchange.

According to the developed methodology of joint representation of ontology and distributed knowledge base, the ontology and IDLS distributed knowledge base are structured as it is shown in Fig.1.8. In it, upper level corresponds to the problem ontology that is shared component representing its common part used in any applied IDLS. The next level represents *shared ontology* specifying particular application of IDLS. Exactly this level of ontology must be developed in such a way that resolves the problems mentioned in section 1.6, which arise due to distribution and heterogeneity of data and respective components of KB. The next level of ontology specifies mostly "*private*" notions of particular source-based components of IDLS.

Thus, the overall structure of IDLS KB and decision making software consists of ontology and distributed KB components, which support decision making procedures performed by particular base–

[1] Here and below if we say "*application ontology*" we mean an ontology of a concrete application from the intrusion detection and intrusion detection learning domain.

and meta–classifiers structured according to *DF meta–model*. Let us remind that *DF meta–model*, in turn, consists of classification tree (in the upper level) and the set of *decision trees* corresponding to each node of the former.

Using of the ontology-based approach to IDLS knowledge base representation resulted in one more untypical problem to be resolved within IDLS. This problem is caused by the fact that application ontology notions are specified in terms of ontology language (in the Project–in terms of the XML language). However, the ontology notion instances (interpretations) are represented in a database language. To provide interaction of ontology and databases of sources (accessibility of data requested in ontology terms), a special gateway destined for transformation of queries to data base from XML into SQL language is to be developed.

A more detailed description of the conceptual model of ontology, its interconnection with distributed knowledge base and description of the algorithms intended for ontology-based resolution of data non-congruency problem was given in [InterRep#2].

1.7.4. Data Mining and Knowledge Discovery Techniques used for Engineering of Distributed Knowledge Bases and Decision Making Mechanisms of IDS

Several techniques for data mining and knowledge discovery are used in the developed IDLS technology. Their detailed description and also specialization (except technique for temporal data mining) were considered in detail in [InterRep#2], and that is why this section intends only to remind them. All the techniques described below are implemented as the classes of the library of training and testing methods of IDLS. In it, five different techniques are used. They are destined for extraction of production rules and association rules from relational and transactional databases which attributes are numerical, binary and categorical and also for mining of temporal data (sequences of packet headers and others). Let us briefly remind them.

Visual Analytical Mining (*VAM*) is a technique, which is destined for extraction of production rules and/or decision trees from databases containing attributes represented in numerical and linear ordered measurement scales ([Gorodetski *et al*-02f**],** [Gorodetski *et al*-00]). This technique is appropriately effective. In particular, it makes it possible to extract production rules specified in terms of the first order logic fragment which does not use quantifiers.

GK2 is a technique for extraction of production rules from data represented in discrete scales (binary, categorical, integer and linear-ordered) ([Gorodetski *et al*-96], [Gorodetski *et al*-02e]). This technique is conceptually close to the well known AQ technique [Michalski-83], but uses different algorithm for extraction of minimal rules.

FP-grows algorithm is a technique destined for association rule mining [Han *et al*-01]. It is well known within data mining and knowledge discovery community as very efficient one. Its formal and informal description can be found in [Han *et al*-01].

An approach developed and used in this Project for *temporal data mining* is of statistical nature and is based on statistical properties of the temporal vector-wise sequences of binary and/or numerical data. This method is described in section 1.6.5 in detail.

The methods of extracting rules from data described above enable engineering of the knowledge necessary for the decision making by base classifiers. Let us consider the question how the base classifiers use the rules for producing of decisions in respect to new incoming data.

According to the adopted IF methodology, in each node of the *classification tree*, and accordingly, in each node of *decision making tree,* the task of binary classification is being solved. Let us designate classes (meta–classes) that correspond to any given node of classification tree as Q and $\overline{Q}$. The production rule mining techniques are formed two sets of rules $\{R_1^+, R_2^+, ..., R_k^+\}$ and $\{R_1^-, R_2^-, ..., R_l^-\}$, where the first set is such that it "argues in favor" of class Q, i.e. contains rules of the type $R_i^+ = F_i^+ \supset Q$, *i=1,2,...,k*, and the second set of rules "argues in favor" of class $\overline{Q}$, i.e. contains rules of the type $R_i^- = F_j^- \supset \overline{Q}$, *i=1,2,...,l.* For brevity, these rules will be called *arguments* hereinafter. Based on the training and testing datasets, a four cell *confusion matrix* of probabilities of correct and incorrect classification, and consequently, a numerical value of a metric adopted to assess the accuracy

of the rule, can be set up in correspondence to each of the arguments. The choice of a metric is application requirement dependent.

Several decision making schemes can be used for the above variant of knowledge base structure. The first and relatively simple variant of decision making consists in counting the weights of the "positive" and the "negative" arguments in favor of one and the other decision (here, the values of the metric that determines accuracy of the rules calculated for the rules, are used as weights), and the conclusion is made in favor of that of the classes whose arguments are "stronger". Essentially, this variant of decision making constitutes a well-known method called "*weighted voting*".

Another, more promising variant is based on the use of probabilistic dependencies between the "*pro*" and "*contra*" rules of the class *Q*. These probabilistic dependencies (calculated, for example, as joint probabilities of different subsets of arguments– pairs of arguments, triplets of arguments, etc. on the training and testing datasets) may be formalized in terms of Algebraic Bayesian Network [Gorodetski-92], which constitutes one of the ways of representing data with uncertainty. Further, for the decision making, the Bayesian inference in the Algebraic Bayesian Network can be effectively used for estimation of the classes' "a posteriori" probabilities. Unlike the first approach, which implicitly assumes the independence of the rules of the base classifier knowledge base, this approach utilizes the existing dependencies explicitly, which secures against adding up weights of rules that correlate very closely and essentially duplicate one another.

The decision making method based on rules that has been implemented can be characterized as a method based on argumentation that utilizes the ideas of *Inferential Theory of Learning* developed by R.Michalski [Michalski *et al*-93]. This theory views learning as a knowledge mining through *knowledge space transformation*. From such a viewpoint, each hypothesis generated by an inductive learning procedure can be considered as twofold. On the one hand, such a hypothesis can be considered as a new generalized attribute specifying the data of a category and forming a new dimension of this data specification. The set of such hypotheses, in turn, can be considered as a new specification of the representation space determined by the primary set of attributes, specifying situation to be classified. On the other hand, a new hypothesis (for example, a rule) may represent a decision procedure that is supposed to be used to discriminate the situation of one category from those of the other ones. Thus, in this case, the set of hypotheses can be viewed as a decision structure [Michalski *et al*-97].

In the implemented model of decision making the rules (arguments) extracted by base classifiers are considered as a new specification of the representation space determined by the primary set of attribute. All of these "new attributes" are binary. For these attributes training and testing datasets are computed from the primary dataset. These datasets are subsequently used for training and testing of base classifiers. As a rule, this step results in extraction more "strong" arguments "pro" and "contra" of the class *Q*. Experience proved that in any case, this process can be repeated up to the situation in which the decision making procedure is expressed by a single rule given over truth values of the lower level attributes and decision is determined by its truth value for an instance under classification. From theoretical point of view, such a procedure uses specification of knowledge in terms of a higher-level predicate logic.

1.7.5. Temporal Data Mining for Anomaly Detection

It is well known that temporal data in itself is a very important and informative source of information and knowledge needed for reliable detection of anomalies in user behavior and/or some kind of intrusions in computer networks. It also plays an important role as a source of information and/or alerts supplementing other data sources used in computer and information security systems. According to the accepted in the Project taxonomy of data sources used in computer network security systems, temporal data are presented at each level that are

(1) Network-based level, where it specifies the sequence of packets' headers associated with each particular connection;

(2) Host-based level, where they, for example, specifies sequences of operating system calls, and

(3) Application-based level.

It is also well known that different types of users' suspicious and abnormal activities are showed within different subsets of data sources used in computer network security systems, can be of different

structure and have different duration. Respectively, detection of the status of the current connection implies combining knowledge, alerts and particular decisions. Exactly this idea forms the basis of intrusion detection technology being developed within this Project

It is important to note that temporal data mining and knowledge discovery in itself and also as applied to intrusion detection is currently not well developed scientific area although it is common understanding in the respective scientific community that the future in this area belong to the behavior-based detection technologies [Stolfo-03].

The most of the existing approaches to temporal data mining exploit statistical basis. An approach used in this Project[1] is also of statistical nature and statistical properties of the temporal vector-wise sequences of binary and/or numerical data. In the context of the Project, development of temporal data mining and knowledge discovery approaches is not a subject of the research and that is why it is below described mostly conceptually with limited formal details because it aims only to provide readers with general understanding of its idea. This approach is described as applied to mining of temporal sequences (streams) of packets headers of current connection. Thus, this data together with data of *Tcpdumps* corresponds to network-based level.

An algorithm implementing the approach described below is included into the library of training and testing methods of MAS prototype destined for learning of intrusion.

Let time be a sequence of increasing integers *1,2* …, and a discrete vector-wise stream of temporal data is observed over this time, i.e.

$$X(1) = X_1, ..., X(k) = X_k, ...$$

Specifically, this data stream is a temporal sequence of headers of packets. Each component of the vector X is *associated with* a name of flag from the ordered set {*URG, ACK, PSH, RST, SIN, FIN*}. The flag can be present or absent in the respective position of vector X and its dimension is equal to 6. Components of the vector X take values from the set {*0,1*}, at that if a flag is present in the packet header then respective component of binary vector X takes value "*1*", otherwise–"*0*".

Sequence of Packet Headers (*SPH*) can correspond to a normal connection or to abnormal one. For example, it can correspond to an attack (known or unknown). It is noteworthy to note that the above task is called as "*anomaly detection*" and exactly it is the subject of investigation within this section. It is a binary classification task in which one class corresponds to a normal user behavior ("normal connection") and the alternative is "abnormal' including suspicious behavior stimulating alert and attacks of any kind.

The basic idea of the developed approach is explained in Fig.1.9. In it, two simple functions given over finite time interval are presented: sinusoid and linear function. Let us suppose that both of them are approximated in two ways: in the first one the Fourier's series are used and in the second one – Taylor's series. It is naturally, that a sinusoid-like function can be precisely approximated by no more than two series members, but in case of use for approximation Taylor's series an infinite number of members are necessary to use. Foe linear-like function the situation is opposite: it would be enough only no more than two members for its exact approximation by Taylor's series and infinite number of members of Fourier's series.

Let us suppose that it is necessary to classify two groups of functions: (1) functions that are representable with given accuracy by linear combinations of n sinusoids of multiple spectrum, and (2) functions that are representable with given accuracy by polynomials of degree n. Two simple approaches can be used to solve the task under consideration. In the first of them, the input function is approximated by Fourier's series based on given finite set of sinusoids of multiple spectrums and the resulting discrepancy is compared with some predefined threshold. If the above discrepancy is upper than threshold then the input function is classified a sinusoid-like, otherwise it is classified as linear-like function. In the second approach one can use linear approximation of input function with a decision rule that is similar to the one described for the first approach.

The main conceptually important conclusion from the above consideration is as follows. If for a given class of data streams an optimal finite functional basis is built and it is other than such a basis for other class of function, then the difference in optimal functional bases can be used to discern

[1] The basic ideas of this approach were developed by the Report authors several years ago.

functions of one class from functions of other one. Actually, the sequences of the firs class will at average be more precisely approximated in terms of its optimal functional basis than in terms of the functional basis of alternative class of functions.

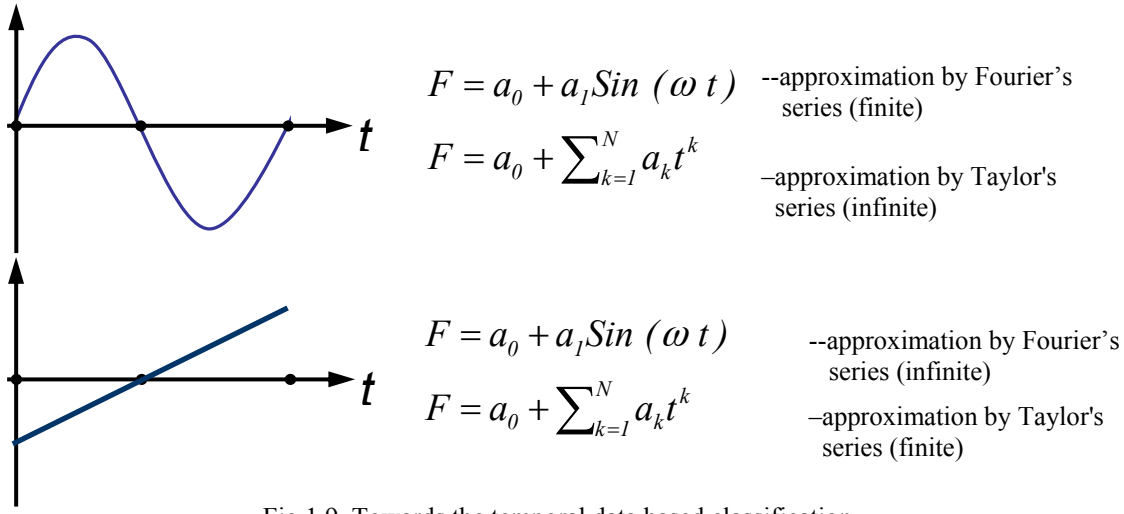


Fig.1.9. Towards the temporal data based classification

Thus, if each class of sequences is approximated by use of its specific (optimal in some sense) functional basis then a threshold-like criterion can be used for binary classification of sequences belonging to given finite set of classes. If the sequences belong to a Hilbert space (in it the norm is defined as inner product) or in its particular case, Euclidean space that it is reasonable a full-range orthogonal basis to select, since this selection simplifies computation of such an approximation coefficients and gives birth new types of classification approaches.

Thus, for practical use the above described approach to temporal data classification and respective learning algorithms, the following tasks have to be solved:

(1) To compute finite set of "optimal orthogonal basis functions" corresponding to normal behavior of users. Such a basis functions has to provide at average a higher accuracy of approximation of the data streams corresponding to normal connections as compared with the approximation accuracy of data streams specifying abnormal accuracy.

(2) To determine a justified value of the classification threshold.

(3) To build an algorithm for approximation of the input data sequences (stream of packet headers) on the basis of the selected (computed) functional basis and also to build an algorithm of discrepancy computing.

(4) To introduce a classification rule, for example, in terms of threshold-like classification criterion, which performs classification on the basis of "*Self*"–"*Non-self*" principle.

The first and second tasks constitute the essence of the anomaly detection learning procedure, and the third and fourth ones provide classification rule design.

It should be noted that although the above described classification approach intends binary classification, after slight adaptation it can also be used in multiple classification tasks. For such an adaptation it is necessary to exploit an idea of binary classification tree performing pair-wise (binary) classification in each node. Other variant of adaptation is computation of particular optimal basis functions for each class of data streams.

Let us consider the developed formal model of temporal data mining for anomaly detection and detection algorithm.

Let {*Normal*} be training and testing sample of binary sequences interpreted as belonging to "*Normal*" connections (user behavior). Due to diversity of users behavior classified as "*Normal*" the sample {*Normal*} can be considered as a subset of instances of vector-wise random sequences. Denote as *{Abnormal}* a sample of vector-wise instances of random sequences corresponding to abnormal connections, i.e. corresponding to *abnormal* users' behaviors.

Below a simplest statistical model of the sample {*Normal*} is considered. In it, the sample of given class of vector-wise random sequences are specified by its mathematical expectation vector $M[X_i]=\overline{X}_i$ depending on time *i=1,2,…* and also by mutual covariance matrix $W(X_i, X_j)=W_{i,j}=M[(X_i-\overline{X}_i)(X_i-\overline{X}_i)^T],\ i,j\in 1, 2, 3$……(indexes *i, j* correspond to time points).

A peculiarity of the anomaly detection task is that it is considered within each particular connection. In general case classification of a dynamic process (in particular, discrete sequence) is performed on the basis of a metric which evaluate a measure of proximity to or similarity with the class etalon model and the classification procedure as a rule exploits a threshold algorithm. Such a metrics can also be presented in terms of inner product of the input process and etalon process of some class or have a sense of conditional probability, etc.

For classification of dynamic processes, it is useful to involve some ideas from theory of signal determination, in which a metric is also used for estimation of the probabilities of the "loss of signal" (*false negative*) and "false alarm" (*false positive*). In anomaly detection task it is very important to form an alert as earlier as possible to prevent negative consequences of attacks. A prognosis of input sequences makes it possible to shorten a delay in anomaly detection.

Let us consider briefly formal aspects of temporal data mining and knowledge discovery used in the implemented prototype for anomaly detection while omitting any proofs. Let us note that learning and classification algorithms described below differ slightly from algorithms implemented in the software prototype of multi-agent intrusion detection system although the basic ideas are the same in both cases. A reason of the above difference is that some theoretical changes were made based on experimental results.

Let us represent a discrete random process $X_1,\ X_2,\ X_3,\ldots$by its approximation which uses linear combination of given finite set of functions:

$$X_i-\overline{X}_i=\left[A_1^T f_1(t_i)\quad A_2^T f_2(t_i)\quad \ldots\quad A_n^T f_n(t_i)\right]^T=\begin{bmatrix} f_1(t_i)^T A_1 \\ f_2(t_i)^T A_2 \\ \ldots \\ f_n(t_i)^T A_n \end{bmatrix}, \qquad (1.1)$$

where $A_k=[a_1^k, a_2^k, \ldots, a_s^k]^T$ – vector of the coefficients of the sequence approximation,[1], $f_k(t_i)$ – vector-wise functions, $s\le n$, *n* is the dimensionality of the vector *X*.

Let vector-wise functions $f_k(t_i)$ are orthogonal and normalized, i.e.

$$\sum_{i=1}^{N} f_{ks}(t_i) f_{kr}(t_i)=\{1,\ if\ r=s,\ and\ 0\ -otherwise\}$$

The approximation coefficients of an arbitrary vector-wise sequence *X*, if the respective functions are orthogonal and normalized are to be found as follows:

$$A_j=\sum_{i=1}^{N} x_j(i) f_j(i) \qquad (1.2)$$

As it was indicated above, the choice of a set of orthogonal normalized functions for approximation of the instances of a sample of the sequences of a given class must be made via minimization of a similarity or a proximity metrics. In this Project, the mathematical expectation of the root mean square error is used as a metric and a set of such functions is built for the set of sequences of class *{Normal}*. Thus, the selected metrics is as follows:

$$Q=\sum_{i-1}^{N} Q_i,\ Q_i=\sum_{i=1}^{N} M[(X_i-\overline{X}_i)^T(X_i-\overline{X}_i)]. \qquad (1.3)$$

[1] It is supposed that vectors are represented as columns.

(N–the sequence length). In other words, selection of the first approximation function ($f_0(i)$ is equal to $\overline{X}_i$) must be made via minimization of the root mean squire of error for the case if a simple approximation function is used. The second approximation function of the functional basis must be orthogonal to the first one while minimizing the above error, etc.

Let us consider the simplest approximate point-wise algorithm of the computation of the set of optimal vector-wise functions $f_k(i)$, $i \in 1, 2, \ldots N$, $k=1,2,\ldots,s$, which ignores cross correlations of the components of the sequences X_i for different moments of the time. Each such function is computed in a table-wise form which columns are mapped to the time $i \in \{1,2,\ldots,N\}$ [1].

Let us apply to the vector $X_i - \overline{X}_i$ a linear transform at an arbitrary time moment:

$$Y_i = B_i^T (X_i - \overline{X}_i), \tag{1.4}$$

where $B_i = [b_1(i), b_2(i), \ldots, b_n(i)]$–matrix of the size $n \times s$ which columns are $b_1(i), \ldots, b_n(i)$, n–dimensionality of the vector X_i (in our case it is equal to 6), s–the number of columns (the number of approximation members), $Y_i = [y_1(i), y_2(i), \ldots, y_s(i)]^T$, $y_r(i) = b_r^T(i)(X_i - \overline{X}_i)$. At that the transform (1.9) is such that

$$M[y_r(i) y_v(i)] = 0 \text{ при } r \neq v \text{ и } M[y_r(i) y_v(i)] = 1 \text{ при } r = v. \tag{1.5}$$

Let us further cal this transform (point-wise) quasi-orthogonalization of a random process. Formally, the last conditions (1.5) can be represented as follows:

$$b_r^T W_{ii} b_s = 0 \text{ и } b_r^T W_{ii} b_r = 1 \tag{1.6}$$

respectively what in matrix form can be written as follows:

$$B_i^T W_{ii} B_i = E$$

Let us note that if matrix W_{ii} is of the rank $s<n$ then the number of components of the vector Y_i is equal to s, matrix E is of size $s \times s$, and matrix B_i is rectangular of size $n \times s$.

From the last formula the following important ones are entailed:

$$B_i^+ = B_i^T W_{ii}$$
$$(X_i - \overline{X}_i) = (B_i^T)^+ Y_i \tag{1.7}$$

where B_i^+ is pseudo-inverse matrix in respect to matrix B_i [Rao-68].

Let us consider a random value corresponding to a discrepancy of an instance X of the random process and root mean squire error of the latter averaged in time:

$$\Delta = (1/N) \sum_{i=1}^{N} (X_i - \overline{X}_i)^T (X_i - \overline{X}_i) = (1/N) \sum_{i=1}^{N} \Delta_i$$

The average of the above discrepancy is as follows:

$$Q = (1/N) \sum_{i-1}^{N} Q_i = (1/N) \sum_{i=1}^{N} M[(X_i - \overline{X}_i)^T (X_i - \overline{X}_i)]. \tag{1.8}$$

While taking into account formula (1.7) expression for metrics Δ and Q can be presented in terms of components of the vector Y_i

[1] An algorithm which takes into account temporal cross correlations of the vector X components is developed but it is not described here because it have not yet been validated via simulation.

$$\Delta^2 = (1/N) \sum_{i=1}^{N} (X_i - \overline{X}_i)^T (X_i - \overline{X}_i) = (1/N) \sum_{i=1}^{N} Y_i^T B_i^T W_{ii} W_{ii} B_i Y_i$$

and

$$Q = (1/N) \sum_{i-1}^{N} Q_i = (1/N) \sum_{i=1}^{N} M[Y_i^T B_i^T W_{ii} W_{ii} B_i Y_i] \quad (1.9)$$

While taking into account the formula (1.5), the latter one can be re-written as follows:

$$Q = (1/N) \sum_{i=1}^{N} \sum_{j=1}^{n} \lambda_j(i) M[y_j(i) y_j(i)] = (1/N) \sum_{i=1}^{n} \sum_{j=1}^{N} \lambda_j(i) \quad (1.10)$$

and in it

$$\lambda_j(i) = b_j^T(i) W_{ii} W_{ii} b_j(i) \quad (1.11)$$

Let us consider how the matrix B_i can be built. Generally said, it starts from search for $b_1(i)$, $i \in \{1,2,...,N\}$ while maximizing

$$I_1 = \sum_{i=1}^{N} \lambda_1(i) \quad (1.12)$$

(see (1.11) concerning how the values $\lambda_j(i)$ are computed).

In the next step the vector $b_2(i)$ is to be computed using the analogous optimization criteria like (1.12) but with additional constraints representing the orthogonality condition (1.6):

$$b_1^T(i) W_{ii} b_2(i) = 0 \quad (1.13)$$

While continuing such process for $r=1,2...s \le n$, and adding in each next step additional constraints representing the orthogonality condition (1.6) for functions $y_j(i)$ in respect to the columns of matrix B_i computed in the previous steps. The number of such steps cannot be more than the rank of the matrix W_{ii}.

In further part of this subsection the function $y_r(i) = b_r^T(i)(X_i - \overline{X}_i)$ is called as r-th principal component of the metrics (1.9), and $\lambda_r(i)$ is called as its weight in the time point i. The sum

$$I_m = \sum_{i=1}^{N} \sum_{j=1}^{m} \lambda_j(i) \quad (1.14)$$

$m \le s$, is called as main part of the metrics (1.9) of degree m. Let us further describe how the above steps aiming at design the matrices B_i, $i=1,2,...,N$, are to be carried out[1].

It can be proved that if to ignore cross correlations of vector X components for different time points, then the task of computation of any $b_j(i)$, $j=1,2,...,s$ $(s \le n)$ is reduced to the series of eigenvalue and eigenvector tasks formulated for each time point $i=1,2,...,N$. In the case under consideration each last task is equal to the principal component analysis task.

Thus, let $j=1$ and let us solve for each $i=1,2,...,N$ the following principal component analysis task:

$$W_{ii} - \lambda(i) E = 0 \quad (1.15)$$

[1] Let us note that all the below transformations suppose that the matrix W is nonsingular for all values of time $i=1,2,...,N$, although this is not the case for practice. Nevertheless, if this matrix is singular, the derivation way is very analogous with some distinctions mostly concerning more accurate manipulations with pseudo-inverse matrices. The given description makes it clearer the general idea understanding.

Let $\lambda_1(i),\lambda_2(i),...,\lambda_s(i)$ are the eigenvalues (s is equal to the rank of the matrix W_{ii}[1]) and $b_1(i),b_2(i),...,b_s(i)$ are the eigenvector forming the transformation B_i of the vector X at the time $t=i$. This operation is performed for all values of $i=1,2,...,N$ and exactly it is the most complicate and time consuming operation of the temporal data mining used in this Project for behavior-based anomaly detection.

The first practically important conclusion that can be entailed from the previous discussion is as follows. If the formula (1.7) represent in the form

$$X_i - \overline{X}_i = B_i^T (W_{ii})^{1/2} (W_{ii})^{1/2} Y_i, \qquad (1.16)$$

then it looks as approximation of an input sequence with orthogonal set of functions $B_i^T (W_{ii})^{1/2}$ and these functions are optimized according to the criterion (1.3) and ordered in decreased fashion according to their contribution into the potential accuracy of the approximation (1.16) of the vector $X_i - \overline{X}_i$ (see also formulae (1.9)–(1.11)). In approximation (1.16) the coefficients presented by the components of the vector $(W_{ii})^{1/2} Y_i$ are variable and this is an important distinction as compared with the representation (1) under search. Fortunately, the simulation showed that both matrix W_{ii} and vector Y_i for the sequences of the class "*Normal*" are not very variable. While taking into account such properties, it was accepted one more simplification according to which instead of variable vectors $(W_{ii})^{1/2} Y_i$ the permanent vectors are used in the approximation (1.16). In other words, the components of the approximation (1.1) are as follows: $f_k(t_i) = B_i^T (W_{ii})^{1/2}$, and vectors A_j, $j=1,2,...,s$, are the subjects of the computations according to the formulae (1.2).

Thus, while summarizing the above discussion and derivations, the algorithm used in this Project for temporal data mining aiming at *intrusion detection learning* in case of binary classification, is as follows:

1. To find the vectors $\overline{X}_i$ and covariance matrices W_{ii}, and mutual covariance matrices W_{ik} for all i, $k=1,2,...,N$, $k>i$, by use of sample of sequences (headers of packets) {*Normal*}.
2. To build the transformation (1.16) via solving the series of task (1.15) that are like principal component analysis
3. To select value of m, presented in formula (1.14), which makes it to justify the decrease of the dimensionality of the representation space by use of $m \le s$ first columns of the matrix B_i, $i=1,2,...,s$, in formula (1.16) corresponding to its largest values of the sums $\sum_{i=1}^{N} \lambda_r(i)$, $r=1,2,...,m$. The value of m is selected in such a way that provides the rest percentage of the sum $I_{rest} = \sum_{i=1}^{N} \sum_{j=m+1}^{s} \lambda_j(i)$ less than a chosen threshold (say, 5% or other validated on the basis of simulation). The latter makes it possible to speed the intrusion detection[2]. Note that this step can be also interpreted as informative feature extraction. The values of $\lambda_r(i)$ are computed according to formula (1.11).

The anomaly detection algorithm based on processing of the input sequence of the packets headers which uses additionally a multi-dimensional regression for sequence development prognosis is described below. Note that it uses prediction of sequences within time windows of the size 12[3].

Let R is the current number of input packet received.

---

[1] It is admitted that this matrix can be singular and that is why we further use pseudo-inversion instead if inversion.

[2] This value is interpreted as percentage of the approximation error metrics arisen due to ignorance of the less informative members of approximation.

[3] The size N=12 was selected due to the fact that that this size "covers" all anomalies that where included in the training and testing sample of sequences {*Abnormal*}.

1. On the basis of R values of X_i received to the current time, a prognosis of this sequence is computed for the forthcoming vectors X_i, $i=R+1, \ldots,N$:

$$Z_2 = \overline{Z}_2 + W_{21}^Z (W_{11}^Z)^+ (Z_1 - \overline{Z}_1) \tag{1.17}$$

where

$$Z_N = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}, \; W_N = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix},$$

$Z_1 = [X_1^T, X_2^T, ..., X_R^T]^T$ –vector, comprising the packet headers of a particular connection, which have been received to the current time;

$Z_2 = [X_{R+1}^T, X_{R+2}^T, ..., X_N^T]^T$ –the analogous vector computed on the basis of the prognosis (1.17);

WZ_N, WZ_{11}, WZ_{12}, WZ_{21} и WZ_{22} –covariance matrices of the vector Z_N and its components Z_1, Z_2.

Let us again emphasize that vector Z_1 is known and vector Z_2 is estimated on the basis of the multidimensional regression (1.17). In should be also remarked that the components of the vector Z_2 are presented in numerical scale but not in binary since they are averaged and can be interpreted as conditional probabilities of the presence of the respective flags in the packet headers. Use of prognosis is very important because it makes anomaly detection procedure quicker.

2. Let $Z = [Z_1^T, Z_2^T]^T = [X_1^T, X_2^T, ..., X_R^T, X_{R+1}^T, X_{R+2}^T, ..., X_N^T]^T$ be an instance of input sequence with added part computed on the basis of prognosis (17). On the basis of known (due to learning procedure) vector-wise functions $f_k(t_i)$=lines of matrix $B_i^T (W_{ii})^{1/2}$, the vectors A_j, $j=1,2,...,s$, are found according to the formulae (1.2). This procedure results in obtaining of the approximation of input sequence with functional basis of class "*Normal*". Let us denote this approximation by symbol $\widetilde{X}_i$

3. The value of the approximation discrepancy (to be compared with a selected threshold) is computed:

$$\Psi = \sum_{i=1}^{N} (\widetilde{X}_i - X_i)^T (\widetilde{X}_i - X_i) \tag{1.18}$$

4. The decision corresponding to the truth value of the predicate

$$\text{If } \Psi < \overline{\Psi} \text{ then } \textit{Normal} \text{ otherwise "}\textit{Abnormal}\text{"} \tag{1.19}$$

is made.

Let us remind that the threshold $\overline{\Psi}$ is determined via testing procedure. The last predicate can also be added by the possible decision "Alert" if the value of Ψ is increasing and is more that some intermediate threshold that is less than threshold $\overline{\Psi}$.

Simulation results corresponding to the described simplified approach showed that it works good in itself and also while combined in the meta-level with decisions produced on the basis of other sources of data.

Experiments proved that the value of the threshold $\overline{\Psi}$ reasonable to compute for each time $i=1,2,...,N$. In the case study the following algorithm of the threshold calculation is used:

1. To compute the values of Ψ (1.18) on the basis of training sample {*Normal*} for all $i=1,2,...,N$.
2. For each $i=1,2,...,N$. the histogram of the values of Ψ is computed and presented graphically.
3. To select a number of the values of the percentage of the examples covered by the threshold, say, from 75% до 100% interval;

4. Based on the histograms indicated in the item 2 and on the selected grid of the values of the percentage indicated in item 3, the values of the thresholds $\overline{\Psi}$ are computed.
5. The selected values of the threshold $\overline{\Psi}$ are tested by use of the developed classification algorithm and testing sample of the sequences. The results received are used for selection of the values of the thresholds that possesses the best features (probability of correct classification, false negative and false positive).

1.7.6. Techniques for Combining of Decisions

According to the accepted methodology, the selected strategy of DF consists in use of hierarchy of multiple classifiers producing decisions on the basis of particular data sources followed by combing these decisions at the meta- level. Let us briefly consider the existing decision combining approaches and techniques.

The existing approaches to and techniques for decisions combining can be grouped as follows:

1. *Voting* algorithms;
2. *Probability*-based or fuzzy algorithms;
3. Meta-learning algorithms based on *stacked generalization*;
4. Meta-learning algorithms based on *classifiers' competence evaluation*.

Voting methods were developed about twenty years ago and are to date in use since they provide satisfactory accuracy in many applications.

The methods of the *second group* use different uncertainty models like probabilistic (Bayesian model of a posteriori probability assessment, Bayesian networks of different structures), possibility theory-based (like Dempster–Shafer theory of evidences), and also on fuzzy set-based models.

However, at present the data mining and knowledge R&D community practically pays the most attention to the methods of combining decisions that use some knowledge about properties of base-level classifiers. General idea of this group of approaches proposed in [Prodromidis *et al*-99b] is called "*stacked generalization*". The idea is to use the results of classifications (records of labels of classes) produced by base classifiers over a sample of interpreted data instances as dataset for training and testing of meta-classifier. This dataset is called learning meta-data. The latter is used for training meta-classifier to combine decisions of base classifiers in a conventional way. In general, stacked generalization-based methods of decisions combining are effective and still being actively researched. A drawback of this group of methods is their inability to preserve already existing set of classifies unchanged if a new classifier inserted in the classification system [Ting-96].

The basic idea of the *fourth group* of decision combining approaches is that each classification algorithm (in our case–each base classifier) is the most “competent” within a particular region of the representation space. Thus, these methods are based on the *evaluations of classifiers' competences* with regard to each particular record of input data specifying a situation to be classified. The main procedure here is to specify in a way the region of competence of each particular classifier in the attribute space [Merz-97].

The core of the approach proposed in it is that a special procedure "*referee*" is associated with each particular classifier. A responsibility of referee is to assess the competence of the respective classifier with regard to particular input data [Ortega *et al*-01]. To provide the referee with such ability, learning procedure is used. Referee learning is a conventional learning task, which is solved on the basis of the same learning dataset that is used for learning of the respective classifier itself. A specific of the referee learning task is that in it other partition of learning dataset is used. Specifically, the same training and testing data are partitioned into two subsets of positive and negative examples at that a positive example is that for which the classifier produces correct classification, otherwise the example labeled as negative. Thus, in competence-based methods, decision combining consists of two steps: (1) detection of the most competent classifier and (2) selection of the classification produced by the most competent one. Further considerably improved version of the basic method was proposed in the papers [Gorodetski-02] and [Todorovski *et al*-00]. The advantages of the competence-based approach are higher accuracy (as compared with both voting and stacked generalization-based approaches) and also its capability to preserve already existing set of classifies unchanged if a new classifier is inserted in the decision combining model.

Two types of methods discussed in this section, i.e. meta-classification and competence-based methods were used in the Project as decision combining techniques.

1.7.7. Training and Testing Methodology

An assumption accepted in the Project is that all the components of distributed knowledge base and distributed decision making mechanisms are designed through training on the basis of the interpreted datasets (samples). Since the decision making procedure of IDS is organized in two stages according to which (see Fig.1.7) in the first stage base-level classifiers produce their decisions and in the second stage these decisions are combined in the meta-level, training and testing procedures have to be organized according to the same structure. Thus, in the first stage, learning procedures perform training and testing the base classifiers and in the second stage they perform training and testing meta-classifier(s).

Conceptual and algorithmic aspects of the task of the first stage were considered above. Let us consider the *peculiarities of base and meta– classifiers learning* caused by the accepted Meta–model of decision fusion. These peculiarities are as follows:

(1) In some cases *several base classifiers may be required even within a single data source*. This necessity can be caused by large dimensionality of attribute vector of the data source and by heterogeneity of data within a single data source. Use of multiple classifiers within a single data source can make it easier to cope with the computational complexity peculiar to any high dimensional and heterogeneous learning task. Other situation, in which it is reasonable to use several base classifiers within a single data source, corresponds to the case if the size of training and testing dataset is limited. In this case, an increase of decision making accuracy can be achieved through use of several different learning techniques and respectively, several classifiers, which decisions should be combined to increase the quality of classification.

(2) It is necessarily to *use meta-classifier* destined for combining distributed decisions produced by the classifiers of the source-based level. This peculiarity entails the necessity *to reserve a certain part of training and testing datasets for meta–classifier learning*. According to the meta–classification approach chosen in the Project as a strategy of decision combining, the reserved part of training and testing dataset is used for computation of meta–data that, in turn, must be spitted into training and testing parts used accordingly in meta–level. It should be noted that not any dataset is appropriate for use in meta–level. An important requirement to such a dataset is its "completeness" requiring that each instance of this dataset *must not contain missed fragments in all the data sources*.

The above peculiarities lead to the fact that training and testing of different base classifiers both within the same data source and within different data sources are very interdependent. The same is true with regard to interdependence of training and testing procedures of base level and meta–level. The major conclusion entailed from this is that all training and testing procedures must be agreed in a way and coordinated. Since in the Project a multi-agent architecture of IDLS is used, then such an agreement and coordination can be only achieved through agent negotiation according to predefined protocols.

The specific of the task of *meta–classifier learning* (*meta–learning*) is caused by the fact that data sources are distributed and this is why meta–learning is a distributed procedure. Therefore, it must be carried out according to a protocol determining how the distributed software components (agents) interact in meta-learning procedure.

A conceptual explanation of the IDLS learning procedures both in base level and meta–level is given in Fig.1.10. In it, the components performing decision making, the learning components and also the components of the software tool supporting distributed knowledge engineering processes are depicted.

Let us remind that in the Project two option with regard to learning components are admitted: either they can only be used in design stage as the components of the software tool supporting knowledge engineering procedures, or they can be designed as a part of IDS system destined to provide the latter with off-line learning capabilities supporting incremental learning on the basis of experience accumulated during IDS operation. In the left hand part of Fig.1.10 the components of IDL responsible for knowledge engineering procedures are depicted while those components responsible for decision

making are depicted in the right hand part of the figure. In the upper and bottom parts of the Fig.1.10 the components of software tool supporting design of intrusion detection learning and intrusion detection components are depicted.

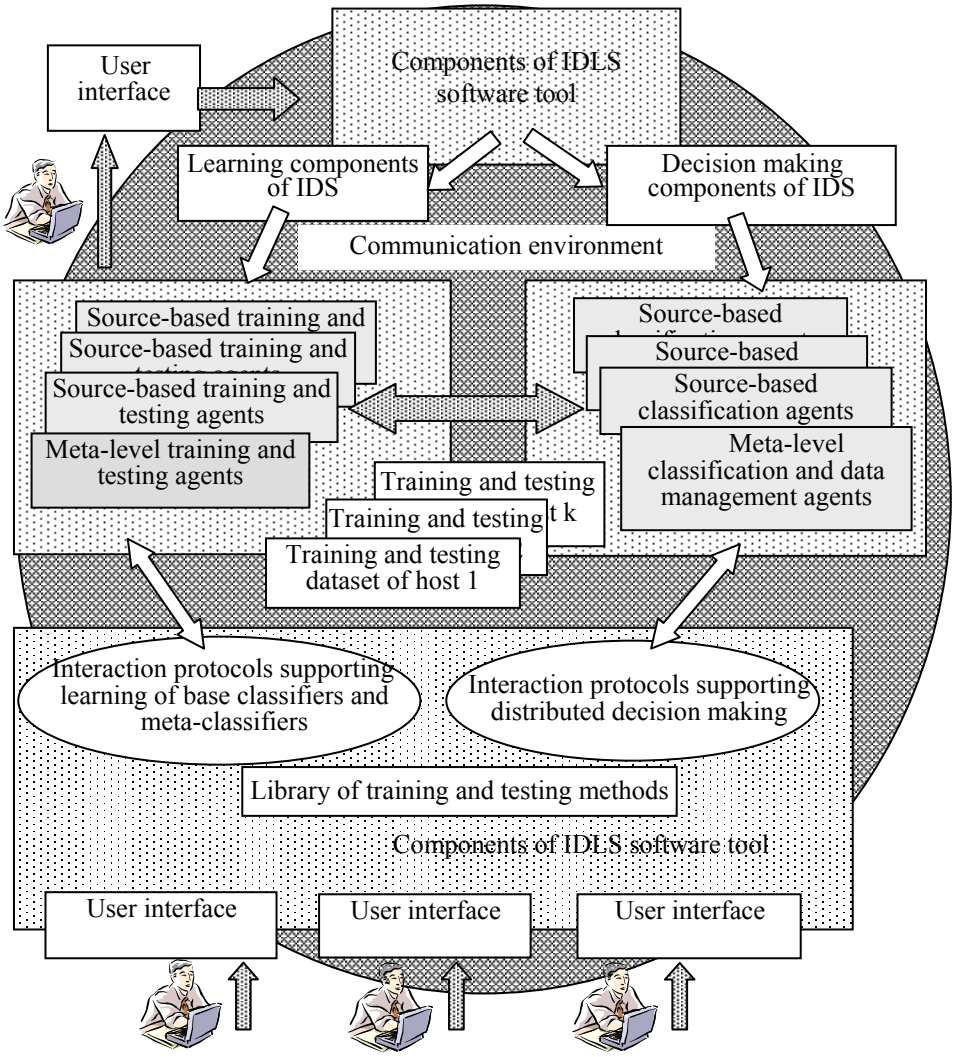


Fig.1.10. Explanation of the interaction of learning and decision making components of IDLS, involved protocols and supporting software tool

1.8. Methodology of Allocation and Management of Training and Testing Datasets

Allocation of training and testing datasets to different components of IDLS to be learnt both in source-based and in meta– levels possesses a number of peculiarities considerably influencing on the whole methodology of learning and subsequently on engineering procedures. These peculiarities are as follows [Samoilov-02]:

- Training and testing datasets allocated to base classifiers and meta–classifiers should be different.
- Training and testing data sets allocated to the same classifier must be definitely different. This difference can be provided in different ways, for example, by use of cross-validation approach.
- Meta–classifier can be trained and tested on the basis of complete samples, i.e. those ones which contain its fragments in all data sources.

Allocation of training and testing datasets to base classifiers for learning needs is also subjected to a number of constraints. This allocation is carried out according to the structure of *DF meta–model*

(see Fig.1.7). Each node of classification tree is mapped a number of preconditions determining properties of training and testing samples. The same also concerns to each node of decision trees mapped to the respective nodes of classification tree. These samples are firstly computed for the nodes of classification tree (according to the respective preconditions) within each data source. Afterwards, the entire data sample mapped to each node of classification tree and split into training and testing sub–samples. The last sub–samples, in turn, have to be used for training and, afterwards, for testing of the classifiers constituting decision tree mapped to the respective classification tree node. Additionally, the above training and testing samples must meet a number of constraints, which are dictated by particular data mining and knowledge discovery techniques used and also requirements of meta-learning technique mentioned in the beginning of this section.

In general case the procedure of allocation and management of training and testing datasets is not as simple as it can seem "at glance" and it is much more complicate than similar procedures peculiar to conventional (non-distributed) learning. In the former process distributed entities that are software agents of IDLS and designers managing learning activity participate and the entire process of training and testing sample allocation is managed by a special protocol coordinating activity of user and operation of the software supporting the technology. This protocol regulates interactions of agents of IDLS and distributed users in computing and allocation of training and testing samples to the respective classifiers.

1.9. Conclusion

The Chapter briefly formulates the main lessons learnt from contemporary studies on data mining for intrusion detection and IDLS prototyping, analyses the main sources of the audit data available for intrusion detection and learning of intrusion detection and describes en engineering methodology developed for design of multi-agent IDLS.

The main lessons learnt from contemporary studies of intrusion detection learning and IDLS prototyping is that it is understood by specialists that ID cannot be efficient and effective if it separately deals with particular data sources. Instead of this, IDS have to be built as an IF systems, which collects traces and evidences of potentially abnormal use of computer network resources and makes decision concerning the status of user's activity on the basis of combining data, information and decisions, obtained from all available sources. Unfortunately, there are very few researches and developments that practically use this understanding and follows IF paradigm. Hence, new IDS architectural solution is very necessary.

This Project is focused on *the development of IDLS components based on use of IF principles and built as multi-agent system*. In this context, the Project is focused on methodology and technology of engineering of multi-agent IDLS based on IF principles and prototyping of a software infrastructure supporting collaborative semi-automated work of designers and software means aiming at design, implementation and deployment of applied IDS with focus on its learning capabilities.

While analyzing data available and informative for intrusion detection and intrusion detection learning, the Chapter presents thorough analysis of such data structures, gives examples of audit data formed in modern operating systems, security systems, and applications. This analysis makes it clear the peculiarities of data to be mined for intrusion detection needs, demonstrates the diversity and heterogeneity of such data structures and specificity of the general distributed data mining and knowledge discovery task in the intrusion detection applications. The Chapter also outlines the representation structures and examples of audit data of different levels of generalization and analyses the most potentially informative characteristics and features of attacks that can be extracted from audit data. Analysis of data available for intrusion detection and intrusion detection learning results in the following conclusions:

- As a rule, information of any single source does not contain reliable evidences needed for timely and efficient detection of attacks and security policy violations. Efficient intrusion detection learning system has to jointly utilize audit data received from multiple sources of different generalization levels (network level, host-based (OS) level, application level).
- The known attack detection learning mechanisms are computationally complex. The complexity can be significantly reduced through the preliminary expert analysis and

identification of the most informative attributes of the audit data. Examples of such attributes are repeated instances and combinations (patterns) of events; mistyped commands; indications of exploitation of the known vulnerabilities, illegal parameters, irregularities in the network traffic parameters and contents; substantial discrepancies in the values of attributes that characterize the system subjects' operations profile, and unexplained problems.

Based on the conceptual analysis of the peculiarities of intrusion detection learning task, the Chapter presents briefly the developed methodology of intrusion detection learning. This methodology has already been presented by its fragments in interim reports ([InterRep#1], [InterRep#2], [InterRep#3]), but as a whole it was formulated in the final phase of the research when the methodology has been tested on the basis of a case study developed. The proposed methodology determines basic principles, methods and techniques of multi-agent intrusion detection learning based on data and information fusion, description of the new problems peculiar to any IF applications and those ones that are specific for IDS applications, proposes possible ways of these problems resolving. The developed methodology also determines requirements to the architecture of IDLS, technology of its engineering and software tool destined for computer support of the technology in the design, implementation and deployment phases.

Chapter 2. Intrusion Detection Learning System Design, Implementation and Deployment. Ontology of Intrusion Detection Learning

Abstract. This Chapter presents the developed and implemented technology of multi-agent Intrusion Detection Learning System (IDLS) design, implementation and deployment. The Chapter outlines general view of multi-agent data and information fusion system technology and divides the engineering processes into two classes supporting (1) design, implementation and deployment of the reusable components of the multi-agent system and (2) design and implementation of data and information fusion-oriented functionalities. Respectively, two software tools intended for support of the engineering processes of the above two classes developed by the Report authors are presented. The first of them called *Multi-Agent System Development Kit* (MASDK) supports design procedures of the first class of technological processes while the second one, *Information Fusion Design Toolkit*, mainly supports design procedures of the second class, i.e. procedures specifically oriented for making use in design of data and information fusion–oriented functionalities. Description of these toolkits, description of the content and peculiarities of design procedures carried out by use each of them are the main subjects of the Chapter. Chapter also describes the developed *ontology* specifying high-level representation of the basic notions of Intrusion Detection Learning domain. The specific of the subject domain under research is that it combines knowledge and therefore, ontologies from three domains, namely, "*Data Fusion and Data Fusion Learning problem ontology", "Intrusion Detection application ontology"* and *"Intrusion Detection Learning application ontology"*. These components of the domain ontology are described.

2.1. MASDK: Generic Model of a Software Agent

The life cycle of knowledge-based multi-agent system (MAS), like any other information system, consists of a number of standard phases: development of a business model and requirements analysis, design, implementation, deployment, testing, maintenance, and evolutionary modification. At present development of software tools supporting the above stages of MAS life circle is a task of great concern. Till now a lot of such software tools has been developed ([AgentBuilder-99], [Bee-gent-00], [JADE-99], [http-1], [http-6]–[http-20]). Between them, the most known ones are such as AgentBuilder [AgentBuilder-99], MadKit [http-6], Bee-gent [Bee-gent-00], FIPA-OS [Posland *et al*-00], JADE [JADE-99], Zeus [Collis *et al*-99], etc. However, despite a large number of research projects of this class, the most of the developed MAS software tools are still in the research stage, but the existing commercial MAS software tools present a very limited functionality.

Most of the developed and being developed MAS software tools use the idea that, despite the diversity of MAS applications and respective implementations, there exist much common (reusable) functionalities in different applied MAS that are practically independent from application to application [Sloman-98]. That is why it is reasonable to implement these common components within a software tool as generic classes and data structures, and reuse them as ready software components in various applications what could decrease the total MAS development time.

This view is also accepted as a basic idea of the developed technology support for IDLS system design and implementation. Practical realization of such a technology support in the form of a software tool supposes to carry out a formidable work to determine reusable components (functionalities, data structures) among many applications. This task was a subject of the research resulted in development of an efficient MAS software tool. Within the developed software technology support the reusable components of MAS are united within a so-called "*generic agent*", which comprises a hierarchy of standard software classes and generic data structures. Actually, generic agent is a nucleus that is "bootstrapped" by the developer via specialization of the software classes and data structures and via replication of software agent classes into instances of particular agents of particular functionalities. These agent instances compose applied MAS. The rest of the MAS development procedure is the conceptual and object oriented design of agent task-oriented collaboration protocols transforming the set of agents into single whole, i.e. into a system. These procedures are supported by a number of special means that play the roles of user-friendly editors aiming at specialization of the

"*generic agent*" to design applied software agent according to the tasks designated to target applied MAS, in this Project, multi-agent IDLS.

A software agent of MAS (the same concerns to agents of multi-agent IDLS) consists of two parts: a *generic agent,* who is also called below an *Invariant platform* for software agent design*,* and an application-oriented component called hereinafter "*applied agent's component*". The former implements the application-invariant functions those are common for all agents of any applied MAS. The latter represents specific application-oriented functionalities, agent's data and knowledge that

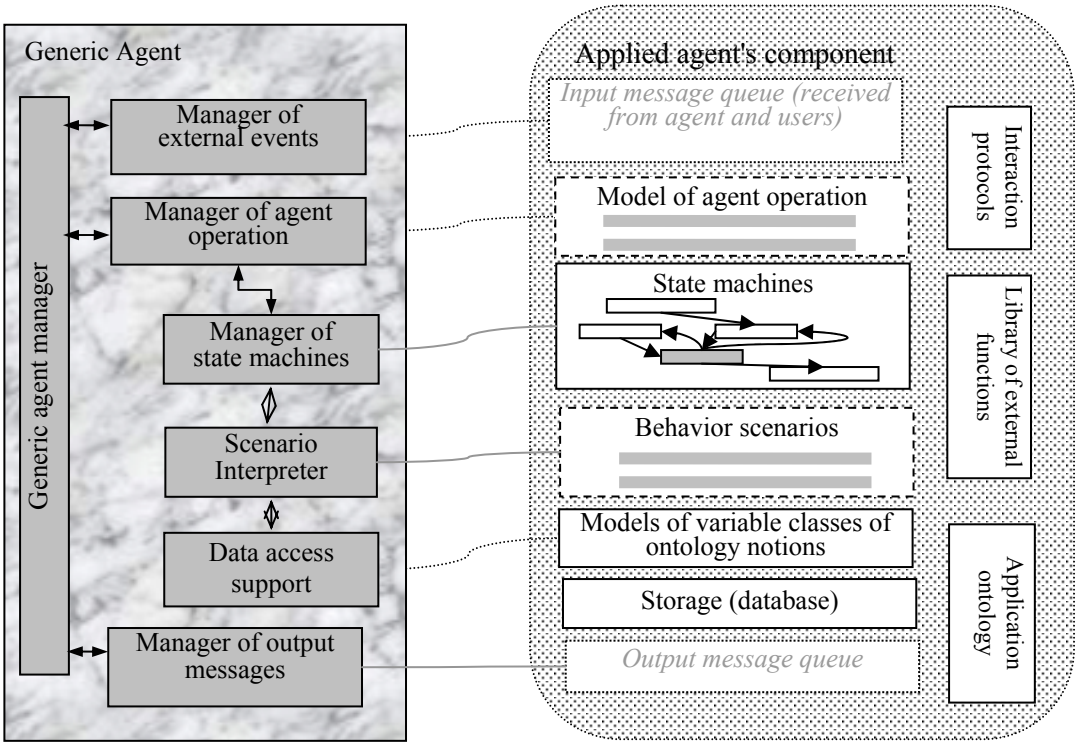


Fig.2.1. Model of *generic agent* and *applied agent's component*

determine the software agent role and the tasks it should solve in a particular application. Fig.2.1 presents conceptual model of *generic* and *applied* agent's components Let us explain it.

A meta-scenario of the agent's behavior is managed by *Generic agent manager*. It manages the allocation of the CPU time between processing the three main execution threads that correspond to the realization of the following functions:

- Primary processing of events incoming from the agent's external world: messages from other agents, user commands and events of the environment perceived by agent's sensors. These functions are performed by the *Manager of external events*.
- Analysis of the agent's current state and management of agent operation depending on the above state and on incoming events. This function is performed by the *Manager of agent operation.*
- Sending messages, which have been formed by the agent according to its behavior scenarios, to other agents according to interaction protocols. This function is performed by the *Manager of output messages.*

The main part of computations is needed for processing of the second execution thread supervised by *Manager of agent operation*. Two other threads are jointly responsible for realization of agent's communication functions including communication with external environment of the agent ("external environment" also includes other agents). In particular, as a result of processing the first execution thread *Manager of external events* forms a specification of the external world, the analysis of which belongs to the responsibility of the *Manager of agent operation*. The task of *Manager of agent*

operation consists in computation of jobs to be designated to *Manager of state machines* in accordance with the management model and the results of the analysis of the current situation. In other words, *Manager of agent operation* initiates the execution of particular behavior functions, or re-starts them if their operation has been broken.

Particular functions of agents (components of its behavior) are represented in terms of *state machines* [Booch *et al*-00]. The function selected for execution is processed by the *Manager of state machines*. It forms the sequence of particular behavior scenarios specified in terms of particular *state machines*. The execution of behavior scenarios is assigned to the program called *Scenario Interpreter.* The program called *Data access support* provides access to the agent's database and modification of the data in database if necessary.

Applied software agent that is an agent instance solving specific tasks within applied MAS, comprises the *generic agent* which model is described above, as well as a number of other components that realize application-oriented functions. The latter components united under the common name "*Applied agent's component*" (Fig.2.1) are as follows:

- *Application ontology* represents the general terminology (classes of notions of application domain and the relationships given over them) used by all agents in messages exchange. The ontology is also used as the terminological basis of agent's knowledge representation. This knowledge is specified in terms of (1) *transition rules* designated to every state of a *state machine*, (2) behavior scenarios provided for by the current state of *state machine*, and (3) behavior scenarios performed in the transition of a *state machine* from its current state to a new state.
- *Interaction protocols* specify agents' interaction in the execution of certain algorithms that are parts of meta-scenarios (executed by interacting agents in coordinated mode. These protocols specify coordination of particular agent behaviors. Protocols are specified in terms of "*roles*" and are performed by agents in accordance with the functionalities supposed by the respective roles.
- *Library of external functions* contains names of methods used for solving specific sub-tasks that are represented as executable code. External functions are invoked in the process of executing the behavior scenarios and fall into the following two categories: *synchronous* and *asynchronous*. The synchronous functions of a scenario are such that while they are executed, no other functions can be executed. Asynchronous functions of a scenario initiated during its execution can be executed in pseudo-parallel mode together with the other functions of the scenario.
- *Input message queue* is a temporary storage for perceived events of the agent's environment (messages received from other agents, perceived events of the environment, user commands). These events are stored in the temporary storage from the moment they are received until the start of their processing by agent.
- *Model of agent operation* contains upper-level specification of agent's behavior. Possible agent behavior scenarios in this specification are represented as a set of particular state machines. *Model of agent operation* specifies rules for choosing functions to execute depending on the current agent's state, and also stores the current states of the state machines if their operation is interrupted.
- *State machines* are used to specify agent behavior meta-scenarios. A state machine specifies the set of states and rules of transition between the states depending on the specified set of conditions and input data.
- *Behavior scenarios* specify behavior (executable functions) of agents in particular states. Agent behavior scenarios are specified for each of the states of a *state machine* and for each existing transition from the current state to a new one.
- *Models of variable classes of ontology notions* are the extension of the specification of object domain ontology. It specifies the storage structure of the ontology notions examples in the *storage* and is used for automatic generation of the structure of that storage, as well as for providing access to data in the process of an agent specialization.
- *Storage* (agent's database) is formed based on the data model (*classes of variable of ontology notions*). It is used to store of the examples of classes of notions and relations which the agent manipulates with. In a general case, *storage* consists of two components used accordingly for temporary and long-term storage of data. *Temporary storage* stores the intermediate data computed

during the current session. *Long-term storage* is meant for storing the initial data and results of agent's operation needed in the future operation. Long-term storage, for example, contains a representation of the initial state of agent's mental model specified at the agent design stage.

- *Output message queue* is a temporary storage of outgoing messages formed by agent. Messages are stored in the queue from the moment they are created until the moment they are sent.

To make it clearer the previous description, let us consider a few typical agent behavior patterns.

- *Initialization of agent behavior functions* on condition that applied multi-agent system is operating. There are three possible ways to actuate the functional behavior of MAS: (1) as a response to an input message from another agent; (2) as a response to an end user's command sent through user interface; (3) as a response to certain events generated by the agent itself.
- *Interaction between agent and end user*. Agents may have or not have a user interface. User interface may be invoked either by a user command from the meta-level of agent management, or by the agent's command generated during executing a certain scenario. The first variant is realized through a user menu that makes it possible for user to generate commands, which actuate the respective dialogs.
- *Response to incoming message*. Upon receiving a message, the agent initiates the execution of the corresponding function (it launches one of its *state machines*). In the process of executing the actions stipulated by the function, the agent can: (1) change the state of its mental model, (2) generate messages addressed to other agents, (3) provide user with data informing him regarding its state.
- *Dialogs*. During operation, agent may initiate dialogs with user or other agents. An outgoing message usually implies a response. Until the receipt of response the operation of agent specified by the corresponding *state machine* is suspended, and that *state machine* goes to stand by mode. This is not applied to the agent's parallel execution of its other functions.
- *Response to user commands*. User command is generated through his/her interface. Agent's behavioral response to user command is identical to response to an incoming message from another agent.
- *Agent's pro-activity*. Agent may initiate its behavior scenarios automatically. This may be a result of an analysis of the current state of agent's mental model. The start of this type of rules may be specified, e.g., based on a time threshold.

2.2. Agent Specification Technology

When the invariant platform for agent design called "*generic agent*" is used, the design of MAS is mainly reduced to specification of classes of agents and their instances, as well as specification of protocols for interaction between agents. Fig.2.2 depicts the main components to be specified according to their functions in the application. Specification of these components and protocols is the essence of the MAS design. Besides the composition of the components specified, Fig.2.2 also shows the partial order given over the set of specification procedures of different components.

Conceptually, these technology consists of several stages. In the *first* stage, application ontology is developed and specified. In the *second* stage, protocols (scenarios) of agent interaction are described and classes of agents are determined. In the specification of interaction protocols, it is necessary to have ontology got ready, because the latter is the mean providing agents with monosemantic understanding of messages they exchange with. Interaction protocols are distributed algorithms executed by a group of agents. In specification of interaction protocols, a notion of *role* is used, which is necessary for specification of functionalities of agents according to designated roles. In the *third* stage, agents of different classes that form the MAS are specified. This stage includes the specification of *Models of variable classes of ontology notions*, *Model of agent operation*, agent's *State machines*, library of *Behavior scenarios,* and *Library of external functions*. Agent of each class can be represented in the MAS by several instances. Specification of instances of each agent class is made in the *fourth* stage of applied MAS development. Instances of agents of the same class differ in names, databases and knowledge bases.

Let us consider some components of specifications technology in more details.

Specification of the Application Ontology

Ontology represents the classes of notions of application domain and relations given over them. Presently, there exists a family of languages used for specification of application ontology, for example, *XML, RDF, DAML+OIL,* etc ([Boumph *et al*-00], [http-2], [http-3], etc.). However, at present these languages are mostly used as means for exchange with ontologies between various applications and developers of various projects. Different projects use their own specialized languages for specification of ontologies as "working" languages. For example, popular ontology editors like *Protégé* ([http-5]), *Ontoedit* ([http-4]), etc. support specification of ontologies in terms of the editor's internal language, while using translating them into *XML, RDF* and some others.

Let us describe the approach to ontology specification that is supported by MASDK:

- *Specification of classes of notions.* A class of notions is specified by name of the class and list of its attributes. Each attribute is specified by name (short and full) and its domain. In some cases, the domain is specified by the domain data type: *{Boolean, string, integer, double}*. In other cases, the specification of the domain also implies assignment the set of admissible values. Class of notions may have both its own attributes specified for this class of notions, and inherited attributes.
- *Specification of classes of structural relationships.* Typical instances of structural relationship classes are "*Part of*" and "*Example of*". Representation of each class of structural relations is usually as follows: (1) Name of relation; (2) Names of notion classes (relation arguments); (3) "Direction" of the relation. Its possible values are *{"neutral", "directed from A to B", "directed from B to A"}*; (4) Arity of relationship. Its possible values are *{"1:1", 1: N", "N: 1", "M: N"}*; (5) Determination of relationship's "*necessity*". Necessity to establish relationship is specified for all classes of notions included in the relationship. If the relationship *<A, B>* is specified as "*necessary*" for class *A* of notions then in the creation of an instance of notion of this class the connection between this instance and the corresponding instance of class *B* has to be specified by the same token. If the relationship is specified as *necessary* for class *B* of notions then in the generation of a notion instance of class *B* the connection between this instance and the corresponding instance of class *A* has to be specified. The latter is used in the generation of database scheme for storing notions instances of the ontology.

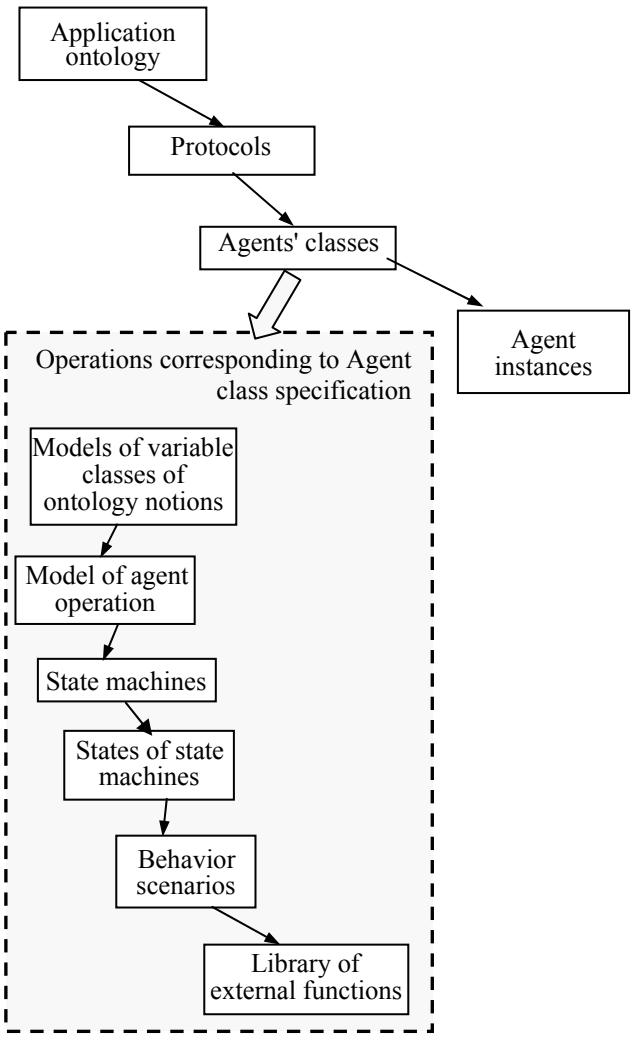


Fig.2.2. Order in which software components of agent classes are specified

- *Specification of inheritance relationship.* In the project only single inheritance is supported that is a class of notions that may not inherit more

than one class of notions. Inheritance relationship between two classes of notion supposes that characteristics of the inherited class of notions become characteristics for the other class of notions that is inheriting from the first one. This relationship is applicable to all attributes of the inherited class of notions, including the ones that would be added to the description of the inherited class and describe its relations with other classes of notions.

Models of Variable Classes of Notions

Models of variable classes of ontology notions are components of application ontology specification. These models are used in solving the following tasks:

- Automatic generation of data storage structure for the dynamic component of agent's mental model.
- Support for data access mechanism.
- Interpretation functions of classes of notions and relationships of ontology in the data storage of the dynamic component of agent's mental model.

In the technology described here application ontology is not only used as shared dictionary of notions to ensure "understanding" between agents, but also as terminology for specification mental models of agents. In the specification of mental model in ontology terms, the agent's behavior scenarios are represented in terms of high-level language, the sentences of which are *on-line* interpreted by components of the "*generic agent*" software. This approach to the mental model implementation makes its operation slower but however, it brings a number of substantial advantages that are as follows:

- It diminishes development and verification cost of software agents because behavior scenarios can be specified by user but not programmer;
- It simpler provides a modifiability of existing agent mental model if necessary.

Models of variable classes of ontology notions specify additional characteristics of application ontology aiming to provide monosematic understanding and processing of messages received from other agents. In particular, these variables describe:

- decomposition of classes of ontology notions into concrete and abstract classes to single out those (specific) classes of notions that have instances;
- requirements for storage of instances of particular notions that can be of two types: long-term or short-term;
- realization schemes for structural relationships.

This kind of information allows for generating the data storage structure automatically. However, in some cases, data storage or some of its parts may be defined from the very beginning. In that case, on the contrary, models of some classes of variables are determined by data storage structures. Example of such application is MAS for data fusion, when interpretations of many ontology notions are represented in the database language which reduces interpretation an *SQL* queries.

Agent Classes

The developed technology uses the notion of *agent classes* in object-oriented style: agent classes correspond to the notion of *agent instances* (agents) just like the notions of *class* and *instance of class* in object-oriented programming. Specifications for data structures and methods of the agent class are inherited by all agents generated as instances of the respective class. Fig.2.3 shows an example of graphic representation of agent classes and their instances as applied to one of Intrusion Detection Learning applications. This example combines two agent classes: *Data source management agent* and *KDD agent of source.* Both classes are used to generate three instances with similar names differing in the names of data sources with which they work.

The agent classes list is determined by the decomposition of applied task and adopted agent interaction protocols typically described in terms of roles. The latter are associated with agent classes. In the particular case, each role defined in the protocol corresponds to particular agent class. For example (Fig.2.3), three instances of agents correspond to the role *Data mining and KDD*. In general case, it is permitted that agents of a certain class may play several roles in one or several interaction protocols.

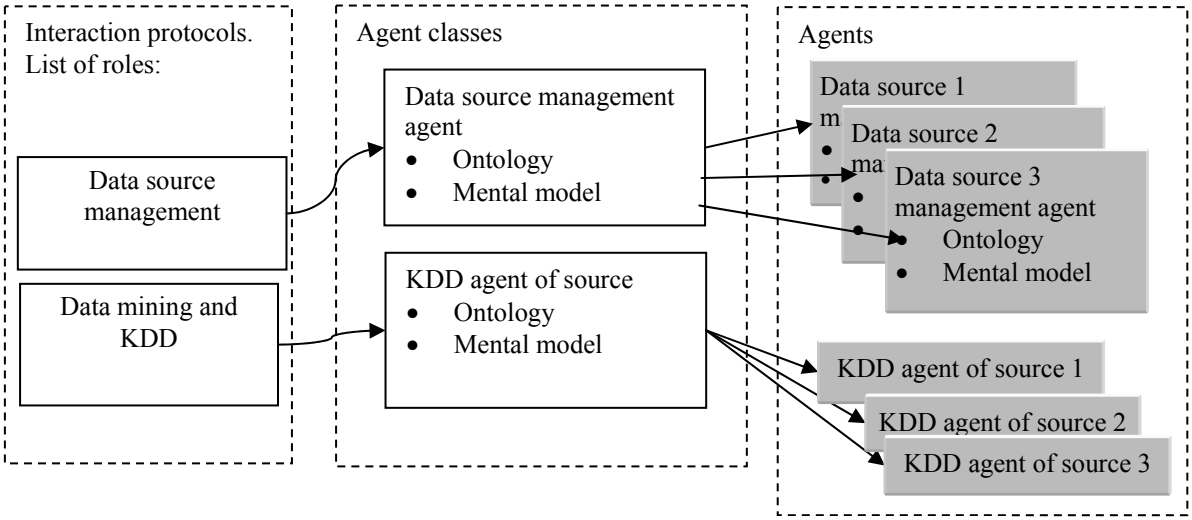


Fig.2.3. Roles, agent classes and agent instances

Allocation of particular functions over agent classes specifies requirements to agent functionality, which in turn serves for creating the agent's mental model. Mental model is the main component of an agent class determining its functionality, behavior, and possibility to carry out this or that role within MAS. Mental model of an agent class includes dynamic and static components. The dynamic component includes a set of facts, or data obtained or generated by the agent during its operation. This data is represented in the data storage, in which the respective variable classes have to be specified. The static component of the mental model contains the specification of management model, state machines, library of agent behavior scenarios, and library of external functions used by the agent for analyzing and modifying the facts of its mental model's dynamic component. List of functions and their allocation over the set of agent classes is, essentially, a conceptual representation of the mental model.

Model of agent operation

Model of agent operation specifies the agent's behavior depending on the current state of environment and its mental model. The set of rules managing agent behavior is specified in the static component of the agent's mental model. The latter has a two-level structure. The upper level specifies meta-scenarios of agent behavior in terms of names of states and transition conditions, as well as names of functions that are associated with states and transitions. Essentially, meta-scenario is determined by the meta-specification of the state machine. At the lower level, a set of agent's particular functions (behavior scenarios) is specified, which interprets names of above actions.

Each state machine, depending on its capabilities, may execute a certain function, provide a service, realize a decision algorithm for a specific sub-task, etc. The set of state machines of a specific agent determines its functional capabilities. Hereinafter, let us call the agent's state machines its functions. It should be emphasized that *all agents of the same class possess identical sets of functions*, with each agent class, as a rule, possessing several functions. The agent's behavior (executed functions and their sequence in time) is either its reaction to events in the environment (messages from other agents, user commands, environment events), or is determined by reaching certain internal states, in which certain functions may be activated without the outside stimuli (through "self-activation"). Model of operation connects these classes of events to the agent's internal state and functions. In particular, it determines, in which situations the agent's functions should be initiated, and under what conditions the agent should resume the action if its execution was interrupted.

Model of operation is explained in Fig.2.4.

The component "*Protocols"* specifies a role or roles assigned to an agent class in different protocols. It contains a representation of protocol dialogs that specify the behavior of agents of this class. If the agent is going to participate in executing a certain protocol through fulfilling a certain role (this is determined by incoming message and the agent's internal state), it uses information about the protocol and the role that is stored in the component *Protocols*. This information, in turn, is used by the agent for initializing the corresponding dialogs. Names of these dialogs, as well as their structure within the interaction protocols, are stored in the component *Protocols*.

In the specification of *Model of agent operation,* the structure of the set of dialogs initiated by the agent in playing a certain role, is recorded in the component *Protocols*. In accordance with the protocol scheme, classes of messages in dialogs are categorized (from the standpoint of agent class) into *Input messages* and *Output messages*. Besides, they are categorized into *Questions* and *Replies.* Input of questions always implies the initialization of a certain response-formulating function; this is why these classes of messages are processed within the component *Initialization.* The processing of the other three classes of messages constitutes execution of functions, and that is why they are

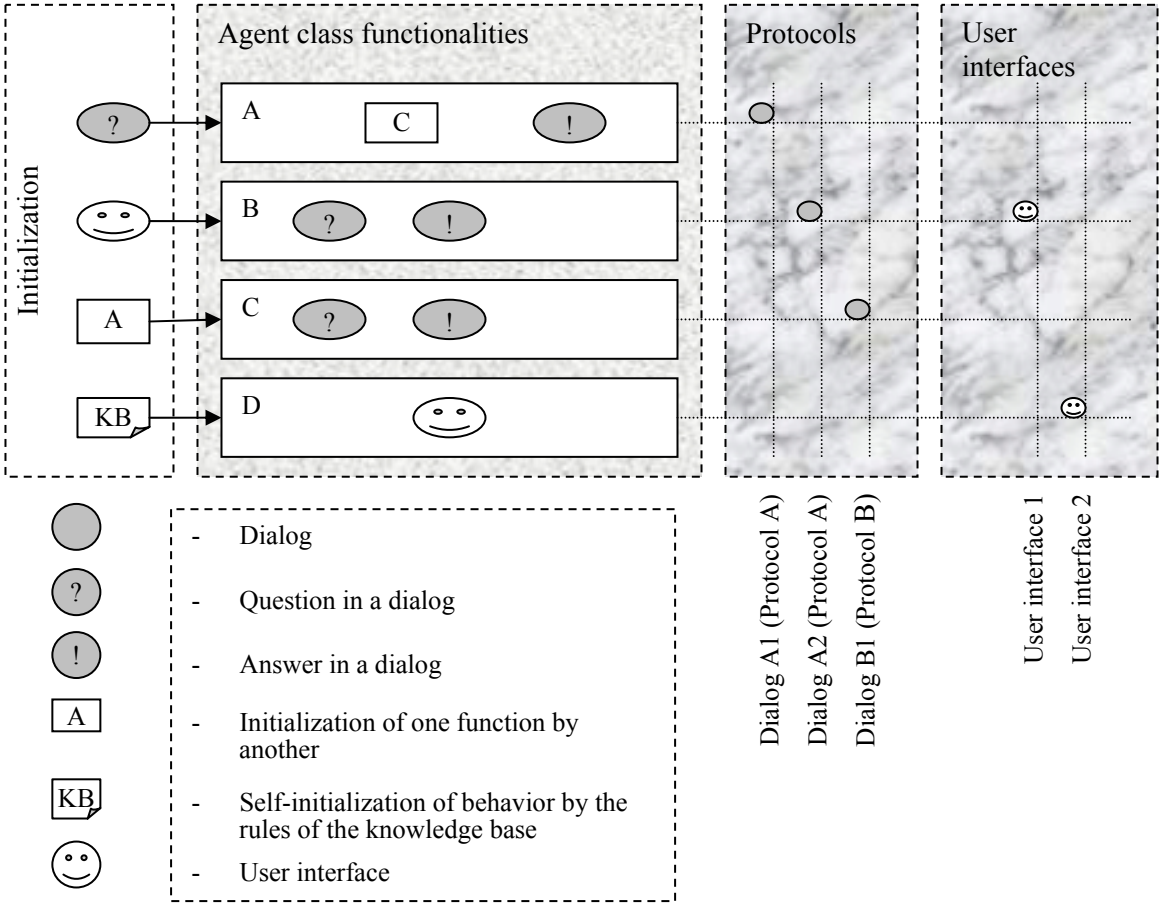


Fig.2.4. Example of a *Model of agent operation*

processed with the functions of component *Agent class functionalities.*

The component "*User interfaces"* specifies the interaction between agent class and users, if such interaction is provided for. This component processes only such user-agent interaction scenarios where user exercises control over the agent. Different user interfaces and functions may be initialized by different components of the operation model. If the interaction is initiated by the user who initializes a certain function, then a corresponding class of events is specified in the component *Initialization.* If the initiative in the interaction lies with the agent, then the initialization of the function is specified in the component *Agent class functionalities.*

The component "*Initialization"* specifies classes of events whose occurrence initiates execution of certain functions. Part of the initialization functions executed by the component has been considered in the description of components "*Protocols*" and "*User interfaces*". For example, a case has been described where it records an incoming message or a user command. Other classes of events that can initiate functions and therefore have to be specified and executed by this component are *Self-initialization rules, Invocation of nested function, Sensor information input.* They are specified in this component of the *Model of agent operation* regardless of the content of other components of the agent class.

State Machines and Behavior Scenarios

State machines used for describing agent classes' functions can also be interpreted as agent behavior meta-scenarios (meta-models). They determine the sequence of execution of particular behavior scenarios associated with the states of *state machines*.

Fig.2.5 shows an example of object-oriented specification of a *state machine* specifying a meta-scenario of behavior of agent aiming at solving a part of the task base classifiers learning that is a component of multi-agent IDLS under development. Conceptually, the scenario for the case if training data are of relational type is as follows. First, user analyzes the quality of rules and coverage of training data by the rules that have already been generated during previous steps of learning. For this purpose, user utilizes the respective user interface. This analysis is resulted in selection of the uncovered examples of training data that are used for learning in the subsequent step. In the next step, the learning procedure analyses columns of training data in order to detect whether there exist equal columns for a pair of features or for rules extracted. (The latter are considered as new features and for each such a rule a new column in training and testing data is generated.) One of equal columns is deleted. If among the features (attributes marking columns) the numerical ones exist then agent transits into the state in which the procedure VAM ([Gorodetski *et al*-02c], [Gorodetski *et al*-00]) is activated. This procedure transforms numerical data into statements about numerical data that are represented in terms of the first order logic, ([Gorodetski *et al*-02c], [Gorodetski *et al*-00]) After transforming all numerical data into predicates of the first order logic, the agent transits into state corresponding to generation of rules for the chosen class of data. In this state, user specifies the parameters of the GK2 algorithm [Gorodetski *et al*-96] and runs it for generated new rules. In the next step, the user analyses the quality of rules according to a given (predefined) criteria and selects those of them that meet the above criteria. Afterwards, the agent again transits into the state, in which it analyzes the whole batch of rules already generated (extracted) using training data. In the subsequent step the agent can continue the search for new rules or transit into one of two concluding state corresponding to the end of the agent's respective state machine operation. These states are (1) end of rule search and adding the generated ones in base classifier knowledge base or (2) temporary stop with the possibility to continue rule generation later.

Specification of a state machine is reduced to specification of a set of states and transitions among them. At that, the set of the state machine states is divided into two classes. States of the first class represent particular agent behavior scenarios, while states of the second class represent dialogs, in which agent exchanges messages with other agents. In the example shown in Fig.2.5, states of the second class are highlighted gray. The difference between the said classes also lies in the fact that the states of the first class are specified at the initial stage of development of the state machine, and the states of the second class are generated automatically based on the specification of the *Model of agent operation.* Besides, the states of the first class require detailed specification, while the states of the second class do not. The only thing that needs to be specified for these states is the transition rules.

The mechanism for processing the states of the second class is invariant to applications and uses data from the interaction protocols specifications. For example, based on the specification of a dialog in the interaction protocol, this mechanism determines the necessity for interrupting the meta-scenario execution (suspending the state machine), if the current dialog needs a response to a certain question. After receiving all the responses, the state processing mechanism re-activates the state machine.

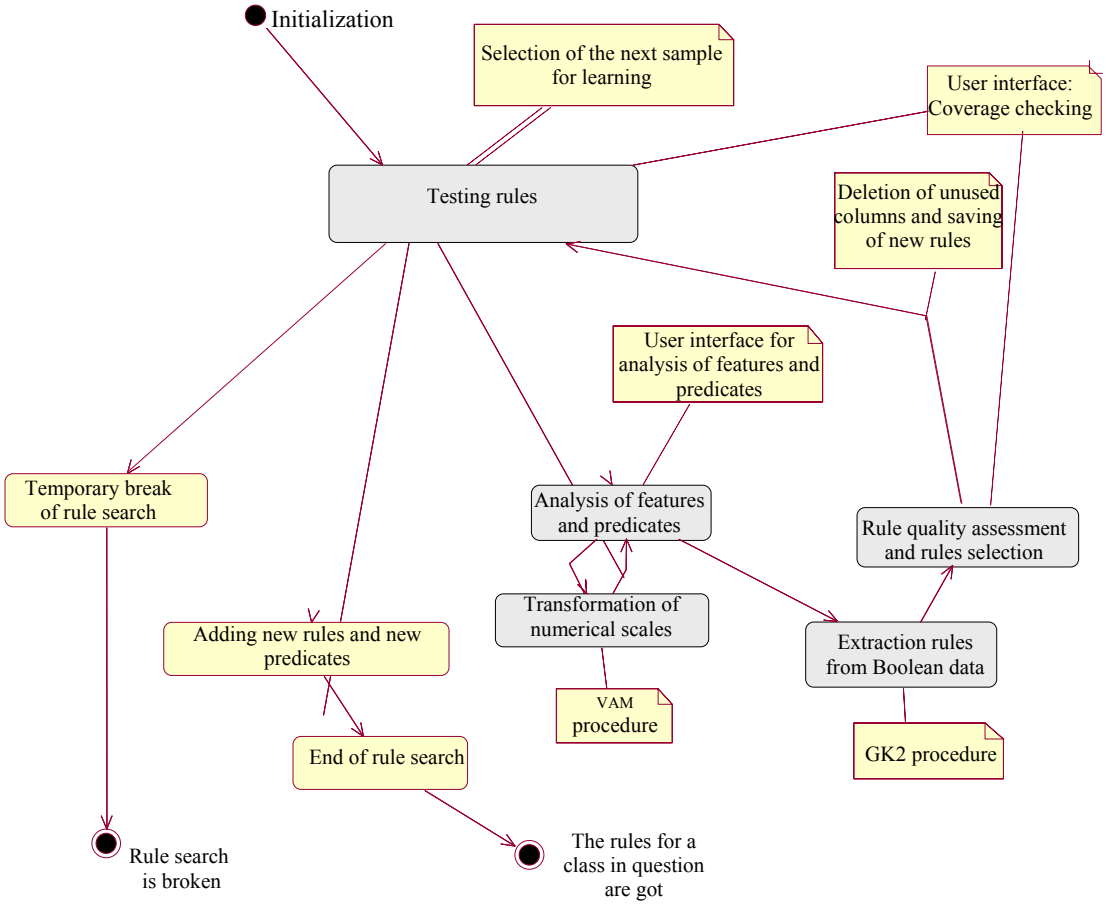


Fig.2.5. Example of state machine specified in UML language

State Machine States and Behavior Scenarios

The states of *state machines* corresponding to the first class indicated in the previous section initiate execution of certain agent behavior scenarios. As the prototype for the model of states of the state machine, the general scheme described in [Booch *et al*-00] was chosen. In a simplified form, the model of a state of the state machine can be represented as two components. The content of the first component is the representation of a particular behavior scenario associated with the state, which is initiated by the machine in that state. The content of the second component is the specification of conditions of transitions from that state.

Behavior scenarios are described with interpreted software code in a higher-level language. In the project, *Visual Basic Script* is mostly used. Another method of representation of behavior scenarios is based on using particular *"if ..., then ..."* rules. In particular, this approach is realized in the intelligent agent development tool *AgentBuilder* [AgentBuilder-99]. However, this approach has a more limited expressive power compared to the approach chosen in this Project.

Behavior scenarios specify the agent's decision-making rules. Through using these rules to analyze the current state of the mental model, the agent performs the necessary actions that can belong to one of six classes. Three of these classes specify the agent's action in the external environment. They are:

- forming messages to be sent to other agents;
- invocation of external functions. Typical examples of components of the latter type are: (1) user interfaces, through which user can both receive information from agent and partially control its behavior; (2) components that realize decision methods for specific, well-formulated tasks, whose description in the scenario description language is either impossible or pointless;

- affecting the environment.

Three other classes of actions manage computations and affect the state of agent's mental model. They are:

- invocation of nested *state machines*;
- invocation of nested behavior scenarios;
- actions modifying the content of agent's mental model.

Invocation of external procedures may be done either in the synchronous or the asynchronous mode. If an external procedure is invoked in the *synchronous* mode, the scenario being executed is paused and re-starts only after external procedure completes its operation. In this mode, data or control commands are entered. In the *asynchronous* mode, the execution of the external procedure and the continuous operation of the state machine take place independently. An example of such operation mode can be the invocation of user interface, through which the user monitors the agent's work without managing it.

Agent Instances and External Environment

Each class of agents can generate an arbitrary number of examples of agents. All characteristics of any agent class are inherited by its instances. Instances of agents also contain additional specific data, such as agent's name, agent's network address, initial state of mental model. Data storage of the agent's mental model can contain initial data specified as early as the description stage. This data is recorded into agent's database.

Specifications of agents' instances and classes of agents are the initial information for the generation of the software agents in the operational environment within a computer network, whose nodes have the *Portal* support service installed. At the system generation stage, it provides remote access to computer. Generation of an agent includes copying the *invariant agent platform*, specification of agent class and generation of the mental model storage, and description of the initial state of the mental model in the storage.

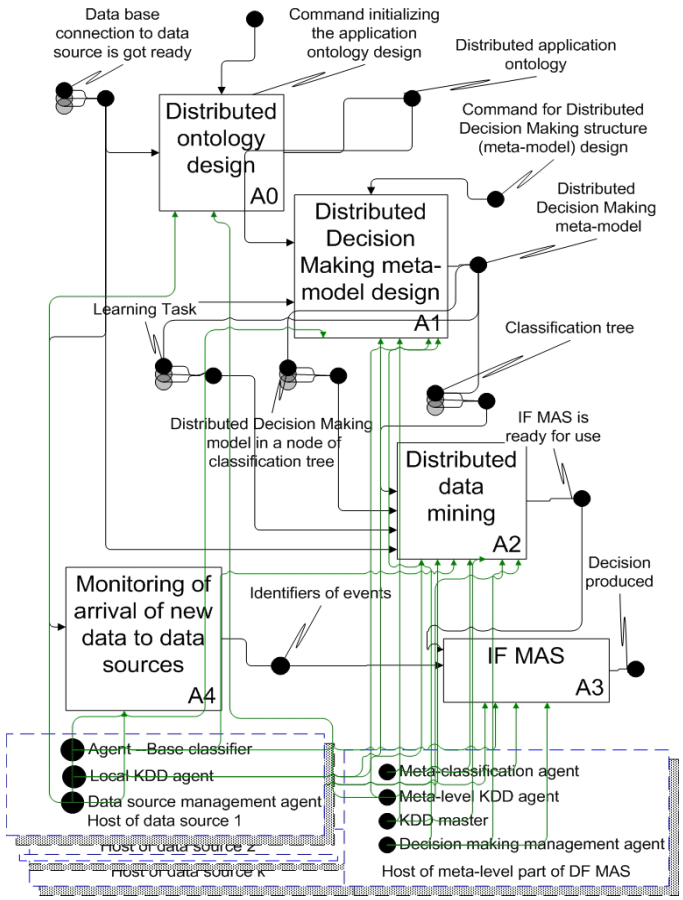


Fig.2.6. High-level protocol of the distributed design of IF MAS

System's operation, besides the *Portal* support service, is also supported by the *Server* of the operational environment providing the operation of certain functions. The most important of them is the remote start of agents (for that, *Server* uses auxiliary program, *Portals*). Agent system can start simultaneously for all agents and selectively based on necessity. Another important function of *Server* is support for agent communication. In particular, all agents' messages go through the *Server*.

2.3. Information Fusion Learning Toolkit

Information Fusion Learning Toolkit is used together with MASDK and primarily supports engineering of the components of an applied IF MAS (in our case - the components of IDLS) that implements data and

information fusion–oriented functions.

The components of *Information Fusion Learning Toolkit* can be divided into three groups that are (1) protocols supporting collaborative design of applications and also collaborative distributed data and information fusion procedures, (2) library of training and testing methods that are used by training and testing agents for learning of IF decision making and decision combining agents, and (3) user interface supporting designers' activity.

It is noteworthy to note, that *Information Fusion Learning Toolkit* actually implements a new kind of agent-oriented software engineering technology that could reasonably be called as "*Agent-mediated software engineering*". As compared with the existing variants of the agent-oriented software engineering, the new features peculiar to the technology supported by *Information Fusion Learning Toolkit* are (1) Support of the *distributed design* of an applied IF MAS that in some cases (private or classified training and testing dataset) is the only admissible one and (2) use of *agents as mediators* of designers in engineering procedures.

A high-level understanding of what kind of technological processes *Information Fusion Learning Toolkit* supports is provided by Fig.2.6. It presents high-level specification of the protocol of activities performed by agent-mediated designers. It should be noted that agents involved in these activities play two roles: they support designers' activity (in training and testing of IF MAS classifiers and meta-classifier) and also participate in operation of the designed IF MAS.

The main engineering activities supported by *Information Fusion Learning Toolkit* mediated by IF MAS agents are as follows:

A0. Distributed application ontology design.

A1. IF meta-model design, i.e. design of decision making tree and classification tree.

A2. Distributed data mining.

A3. Information Fusion (Distributed decision making).

A4. Monitoring of arrival of new data to data sources.

It is noteworthy to note that users collaborate in design procedures according to these protocols. In the most part of the design activities the initiative belongs to users but protocols enforce them to follow a predefined order of an applied IF MAS engineering, provide users with design templates thus monitoring the "design discipline" as a whole. Below in this section the content of the users' and agent-mediators' activity is described.

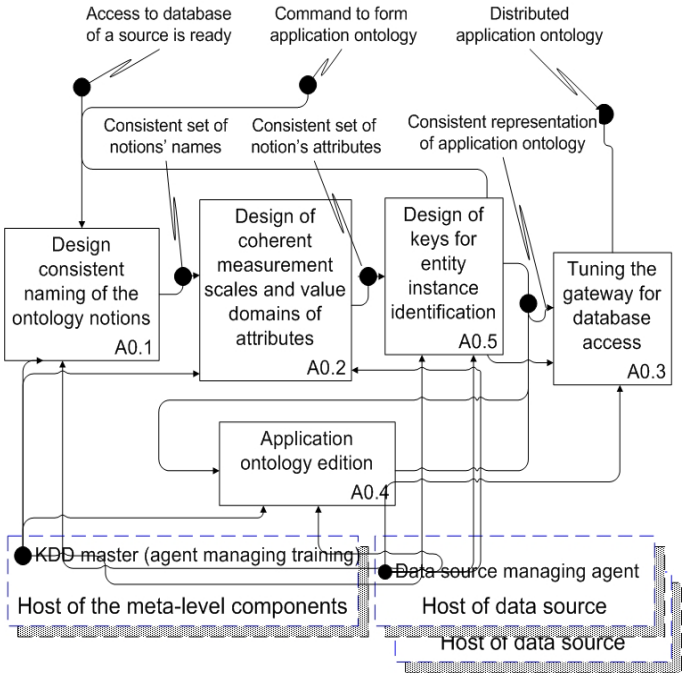


Fig.2.7. Protocol for distributed design of application ontology

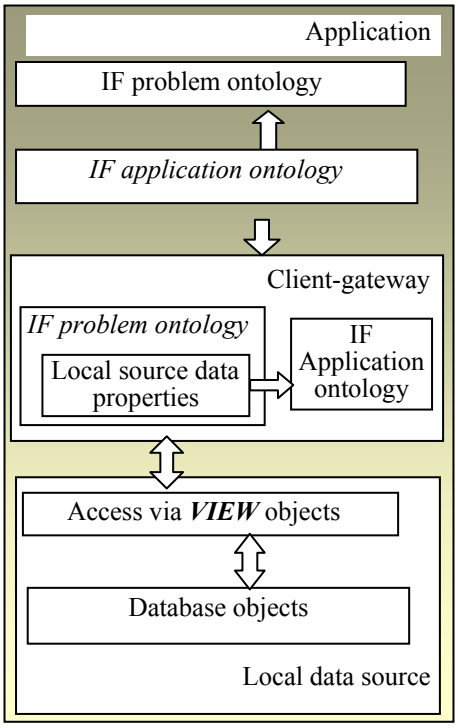


Fig.2.8. Three-level hierarchy of access to the database objects

Distributed ontology design

The key peculiarities of IF systems architecture and its operation come out of the fact that data sources are *distributed*, *heterogeneous* and together constitute a *large scale* data and information processing task. In IF system engineering practice these peculiarities are challenging and considerably influence on many issues of IF MAS *design*. According to the modern view of how to cope with these challenges the ontology-based approach is the most promising one. In the project this approach is in the focus of both IF MAS architecture and design technology.

The responsibility of Information Fusion Learning Toolkit is to specify instances of the application ontology, to maintain ontology consistency on the whole and to provide the ontology under design with a number of the necessary properties considered below.

The main procedures of the ontology design protocol, the agents participating in its execution as well as input, intermediate and output data are presented in Fig.2.7. The particular processes supported by this protocol are as follows:

1. *Design of consistent naming of the ontology notions* providing IF MAS components (agents) with *monosemantic understanding of the terminology* used in formal specification of domain entities and therefore providing mutual understanding and *monosemantic interpretation* of the messages, which the agents exchange with. This procedure maintains the consistency *shared thesaurus* of agents.

2. Design of the *coherent measurement scales and value domains of the ontology notions attributes.* This task is also solved via negotiation of the data source managers and meta-data manager according to a particular sub protocol extending this process.

3. Design of *keys for entity instance identification* intending to solve so-called "*entity instance identification problem*".

4. Tuning the gateway for data base access. This process of application ontology design come out from the fact that application ontology notions are specified in terms of ontology language (XML, etc.) but their instances (interpretations) are represented in database language. To provide interaction of ontology and databases of sources (accessibility of data requested in ontology terms), a special gateway is to be designed. Fig.2.8 explains the content of the fourth process of the protocol depicted in Fig.2.7.

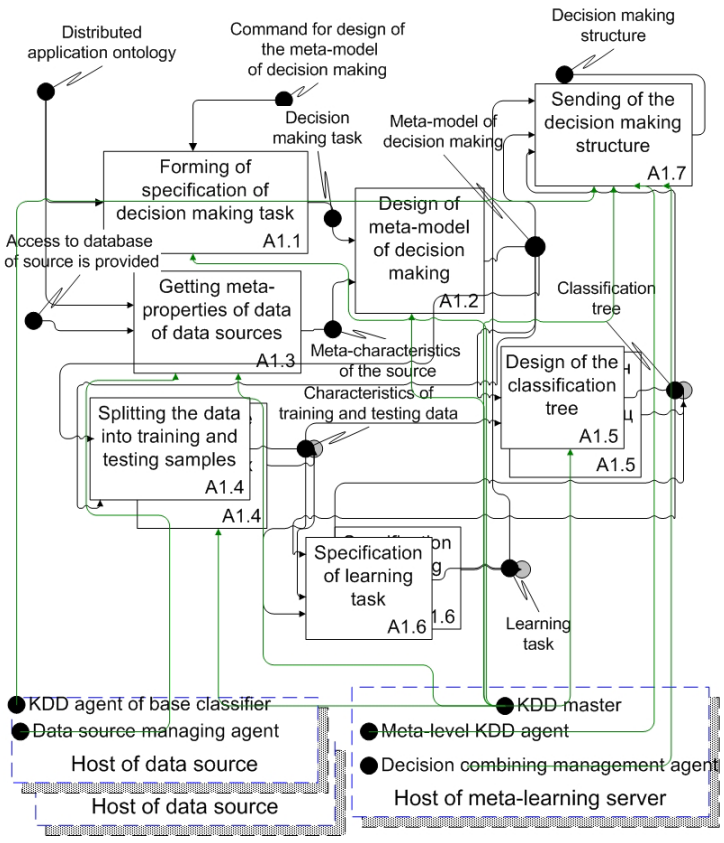


Fig.2.9. Protocol of IF meta–model design

The protocol of distributed design of distributed ontology completely solving the particular problems is developed in detail and implemented as a component of Information Fusion Learning Toolkit.

In the implemented versions of MASDK and Information Fusion Learning Toolkit the applied IF MAS ontology is so far specified in terms of the XML language. In their next versions RDF, DAML+OIL languages will be used.

Information fusion meta–model design

According to the high level protocol (Fig.2.6), design of IF meta–model is carried out as the second step of the technology supported by Information Fusion Learning Toolkit. The upper level of IF meta–model is constituted by classification tree. Each node of this tree corresponding either meta–

class or class of situation is mapped a decision tree, which in turn can comprise several level. The main design procedures, their ordering and agent participating in performance of this or that design procedures of IF meta–model are determined by the protocol presented in Fig.2.9.

Distributed data mining

The protocol of distributed data mining is the core of IF system technology because it conducts training and testing of particular classifiers and also manages decision combining that is one of the basic IF system functionalities. IDEFO diagram of this protocol is presented in Fig.2.10.

Distributed decision making

The methodology of distributed decision making is realized according to the special protocol that is depicted in terms of IDEF0 diagram in Fig.2.11.

2.4. Problem Ontology for Data Fusion and Learning Data Fusion

The conceptual basis of the multilevel Intrusion Detection Learning Problem is formed by three tightly correlated components of the ontology. These components are as follows:

(1) Problem ontology for Data Fusion and Learning Data Fusion;

(2) Intrusion Detection Application ontology;

(3) Intrusion Detection Learning Application ontology.

Ontology specifying data fusion and learning data fusion problem domain represents knowledge of the basic concepts of the problem associated with data fusion and learning data fusion that is independent from a concrete application. The concepts of this ontology reflect knowledge that is common for most of the data fusion, data mining, knowledge discovery and knowledge based decision-making systems. This ontology specifies upper-level knowledge of this problem and can be used in various domains (not only in the subject domain associated with the detection intrusions into computer network and learning such intrusion detection).

The Intrusion Detection Application ontology specifies knowledge of the basic concepts peculiar only to a domain of the intrusion detection.

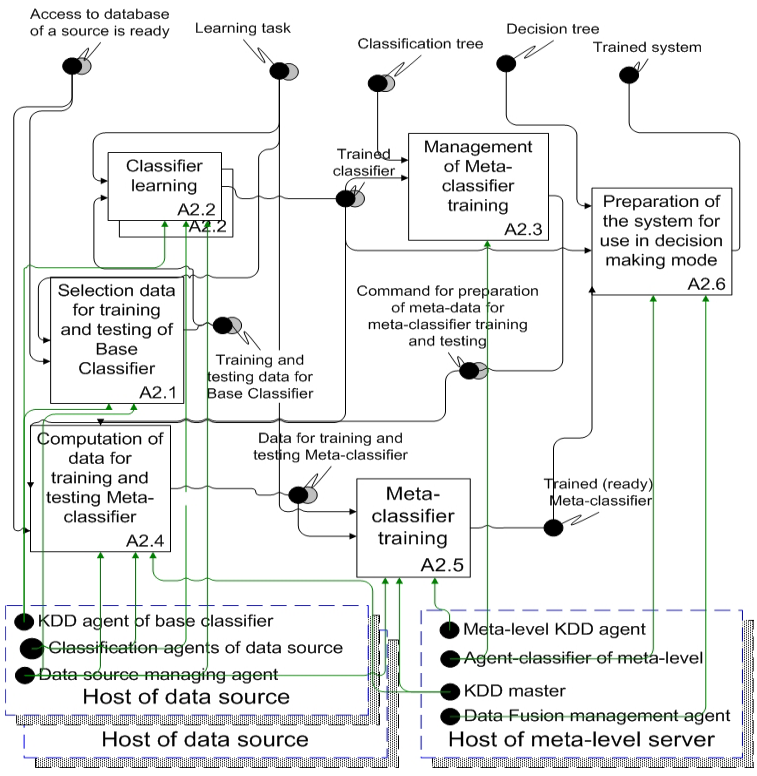


Fig.2.10. Protocol managing distributed data mining procedures

The Intrusion Detection Learning Application ontology sets knowledge of the basic concepts peculiar to the respective subject domain.

In this section, the Problem ontology for Data Fusion and Learning Data Fusion is described briefly [InterRep#1]. The problem domain ontology specified below borrows the basic structure proposed in [Zitkov *et al*-00] and extends the latter in some aspects. *The basic notions of the problem domain ontology for data fusion and learning data fusion problems* are the following: Learning application domain; Basic data; Local data source; Data dictionary; Object; Attribute; Attribute type; Attribute domain; Relation; Object domain entity identifier; Notion; Interpretation function; Base classifier; Meta-data; Meta-classifier; Source meta-

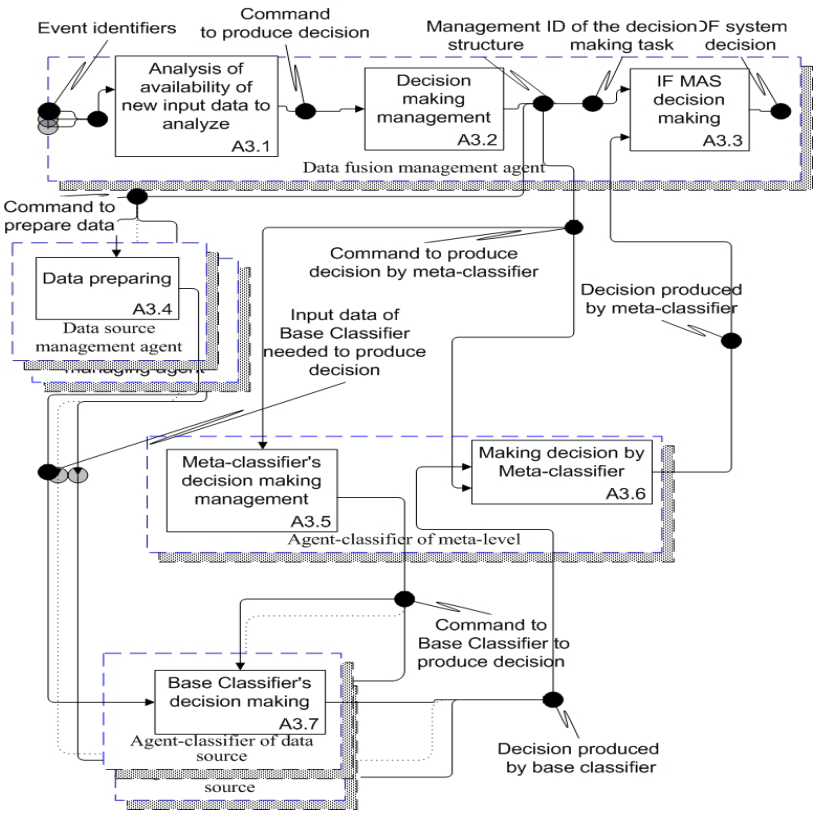


Fig.2.11. Protocol for distributed decision making

characteristics; Sample set; Training data; Testing data; Local discovery task; New knowledge; Feature; Meta-feature; Association rule; Frequent episode; Frequent episode rule; Classification rule.

Below the brief definitions and explanations of the above ontology notions are given. The basic notions are referred using an *italic* letters.

Learning application domain — a real or abstract system existing independently from the discovery system. An application domain consists of (1) objects, which can belong to one or several classes and jointly form the set called universe, (2) specific attributes of objects and (3) relations between objects. Discovery systems attempt to discover domain models and domain theories. In the Project the learning application domain is a domain of intrusion detection into computer networks.

Basic data — real data residing in local data sources in the form in which it is usually gathered. Some local sources may be data warehouses with pre-processed and cleaned data; others may contain direct sensor readings not ready for immediate use. In the Project the basic data corresponds to the *audit data* generated on host, network and additional sources levels.

Local data source — geographically, physically, or logically separate data source with its own data storage structure and mechanism.

Data Dictionary — an ordered set of data about the attribute types and other attribute semantics.

Object (entity, event, unit, case, situation) — an entity or a part of an application domain. Objects belong to the classes of similar objects, such as persons, transactions, locations, events, processes and situations. Objects are characterized by attributes and relations given over objects.

Attribute (field, variable, feature, property) — a single characteristic of an object belonging to an object class. Attribute can be viewed as mapping of objects to values of the given type combined with the mapping of meaningful operations and relations on objects into operations and relations on values.

Attribute Type — a type of values in the attribute domain, the operations and relations over the values, which are meaningful for the objects. An attribute can be binary, nominal, ordinal, interval-valued, and continuous. The value types can be simple (names or numbers, one per object), but the value may be also a complex structure like a time series, temporal sequence, etc.

Attribute Domain — a set of possible values of an attribute.

Relation — a set of object tuples (pairs, etc.), which have specific meaning, interpretation. For example, "event *a* is earlier than event b", i.e. event *a* ("read the directory /ad", etc) occurs earlier in time than event b ("open the file *passwd*"). Relations can also be formally specified as relations over object values.

Object domain entity identifier — an analog of the first key for a flat table defined for an object of the object domain. For every such identifier, a rule is defined within the framework of the problem domain ontology, which can be used to calculate the value of this key. For example, a unique combination of several attributes of a specific entity could be one such rule.

Notion — a function built on the terms of the local problem domain ontology (possibly using the already defined notions).

Interpretation function — a function that establishes relationships over objects of a database (fields, procedures, etc.) and over the attributes of objects or notions of the local problem domain ontology. Interpretation functions have a certain measurement type ("measurement scale") and domain.

Base classifier — a decision support system (classifier) based on rules derived from the single source of data and aiming at making classification using corresponding inference mechanism.

Meta-data — data formed by generalization and joining in a tuple the decisions made by the base classifiers regarding an object specified in terms of local data sources (in terms of common ontology).

Meta-classifier — a base classifier that uses meta-data as input data to be classified.

Source meta-characteristics — characteristics of the local source represented in terms of the local ontology. Such characteristics could be, for example, quantity of objects of such class of the object domain, percentage of missed values of object domain object class attribute, etc.

Sample Set — a subset of objects of an application domain (population) for which data are available or sought. Probabilistic properties of the sample set should be given or assumed, that relate the sample set to the whole domain universe, or weights for objects that indicate their representativeness. Especially for probabilistic domain models, the sample set should be a representative sample of the joint distribution.

Training data — data used for knowledge base learning (in particular, in intrusion detection learning, it is used for generation (modification) of pattern, frequent episodes and rules forming knowledge base of intrusion detection system).

Testing data — data used testing learned knowledge base, for example, intrusion detection knowledge base. They include data not used in the training data.

Local Discovery Task — a request for a specific component of new knowledge. "Find regularity", "generalize regularity", "combine regularities into theory" are examples of such tasks. Each discovery task can be best characterized by the search space explored to accomplish that task, because we do not know in advance the concrete form of new knowledge or even whether any knowledge will be discovered in a given input.

New Basic Knowledge — knowledge augmenting or refining the contents of the current domain model and/or domain theory. New knowledge can be new to the user and extend the user's mental model of the application domain. For an autonomous system, new knowledge may be just knowledge new to the discovery system.

Feature — a field describing the basic data records ([Lee *et al*-98a], [Lee-99]). The feature can describe the characteristics of some event, for example, the source address of the network packet, the number of bytes transferred, etc.

Meta-feature — a field describing the meta-data records or a feature specified as a function having features as arguments. For example the meta-feature can describe average duration of network connections, etc.

Association rule — a rule for deriving multi-feature (multi-attribute) correlation from a set of data (basic data or meta-data). Formally, given a set of records, where each record is a set of items, an association rule is an expression $X \Rightarrow Y$; *confidence; support*. X and Y are subsets of the items in a record, *support* is the percentage of records that contain $X + Y$, whereas *confidence* is *support*$(X+Y)$ / *support*(X) ([Srikant *et al*-95]). In intrusion detection area the order of attributes in patterns X and Y is an important property of the association rule.

Frequent episode — a set of events that occur frequently together within a definite time window or, in other words, a time-based sequence of events frequently encountered together.

Frequent episode rule — a rule for deriving frequent episode. For X and Y where $X+Y$ is a frequent episode, $X \Rightarrow Y$ with *confidence = frequency(X+Y) / frequency(X)* and *support = frequency(X +Y)* is called a *frequent episode rule* ([Mannila *et al*-95], [Lee *et al*-98a], [Lee-99]).

Classification rule — a rule that maps a data item into one of several predefined categories.

Each of the represented concepts specified in more detail at lower levels of the ontology.

2.5. Intrusion Detection Application Ontology

The intrusion detection application ontology represents knowledge specified in terms of the basic notions of the intrusion detection domain. This knowledge is not influenced by the specifics of design and implementation issues like chosen high-level architecture, architecture of the particular learning components, techniques used for attack detection, etc.

The structured representation of the basic notions of the Intrusion Detection domain ontology is as follows ([Cheswick *et al*-94], [Cohen-97], [Howard-97], [Howard *et al*--98], [Krsul-98], [Landwehr-94], [Lee *et al*-98a], [Lee *et al*-98b], [Lee *et al*-99a], [Lindqvist-97]):

- Message traffic
  - Network packets
    - Header
    - Data field
    - Ending
  - System calls
  - Application events
  - Users' commands
- Event
  - Significant event
  - Critical significant event
- Audit records (data)
- Pattern (tag, profile, sample)
  - Significant event
  - Simple sequence of significant events
  - Combined structured sequence of significant events distributed across multiple hosts
- Rule
  - Access control rule
  - Identification and authentication rule
  - Intrusion detection rule
- Activity
  - Authorized activity
  - Non-authorized activity
- Scenario of activity
- Vulnerability
  - Design vulnerability
  - Implementation vulnerability
  - Configuration vulnerability
- Connection
  - Actual connection
    - Sequence of significant events
    - Sequence of patterns or activities
  - Completed connection
    - Sequence of patterns or activities
    - Category (normal/abnormal and, possibly, the name of abnormality class)
- Attack target
  - Logical entity
    - Account
    - Process
    - Data

        - Files
        - Data in transit
    - Physical entity
        - Component
        - Computer
        - Network
- Attacker
- Attack tool
    - Physical attack
    - Information exchange
    - User command
    - Script or program
        - Shell script to exploit a software bug
        - Trojan horse
        - Password cracking program
    - Autonomous agent
        - Virus
        - Worm
    - Toolkit
        - script or program
        - autonomous agent
    - Distributed tool
    - Data tap
- Attack
    - Reconnaissance
        - Identification of hosts
            - Network ping sweeps
            - Port scanning
                - TCP connect scan
                - TCP SYN scan
        - Identification of Services
            - Port scanning
                - TCP connect scan
                - TCP SYN scan
                - TCP FIN scan
                - TCP Xmas Tree scan
                - TCP null scan
                - UDP scan
                - Half scan
                - Scanning “FTP bounce”
                - Dumb host scan
                - “Proxy”- scanning
        - Identification of OS
        - Collection of additional information
        - Resource enumeration
        - Users and groups enumeration
        - Applications and banners enumeration
    - Implantation and threat realization
        - Getting access to resources
            - Direct connection to a shared recourse
            - Installation of backdoor server daemons and trojans
            - Exploitation of known server application vulnerabilities
            - Cracking of PWL file
            - Anonymity access to Ftp server
            - Brute force password guessing
            - Password stealing attack

- Escalating privilege
  - Password cracking (use of John- or L0phtcrack)
  - Use of known exploits (use of Ls_messages, getadmin or sechole)
- Gaining additional data
  - Evaluate trust relations
  - Search for cleartext password
- Threat realization
  - Confidentiality violation (information disclosure) realization
  - Integrity violation (information corruption) realization
  - Availability violation (denial of service) realization
    - SYN flood
    - Land
    - Ping flooding
    - Smurf
    - Ping of Death
    - UDP flooding
    - storm of inquiries to FTP-server
  - Theft of resources realization
- Covering tracks
  - Clear logs
  - Hide tools (Rootkits, File streaming)
- Creating back doors
  - Create rogue user accounts
  - Schedule batch jobs
  - Infect startup files
  - Plant remote control services
  - Install monitoring mechanisms
  - Replace apps with trojans

- Unauthorized result of activity (attack)
  - Increased access
  - Information disclosure
  - Information corruption
  - Denial of service
  - Theft of resources
- Attack (intrusion) detection technique
  - Misuse detection
  - Anomaly detection
- Intrusion detection agent

The shift of the corresponding notion shows dependence (for example, *part of* relation) of a shifted notion from the situated above non-shifted or less-shifted one.

Let us explain a part of notions *of the domain ontology of the intrusion detection in computer networks*. Using *italic* font refers to the basic notions.

Message traffic — a stream of network packets into hosts from inside and/or outside of the network or generated by hosts, and also a stream of *System calls*, *Application events* and *Users' commands* within hosts.

Network packets — messages of the network level. A network packet consists of *Header* (service information, source address, destination address and other fields), *Data field* and *Ending* of a packet (check sum, delimiter).

System call — operating system kernel events.

Application events — applied program events.

Users' commands — instructions generated by the user.

Any message is assigned a set of attributes, which values are possibly assigned probability determining, whether particular attribute is transferred on the legitimate channel, or on the unauthorized access channel, and also whether it belongs to an unauthorized access script. These

attributes comprise two subsets: (1) attributes associated with the properties of the message itself (source, receiver, transfer time, outgoing computer network status, predictable status of the network receiving the message, etc.); (2) attributes assigned by a message processing procedure (a processing time, involved resources, the status, with which it passes the network, etc.).

Event — an action directed at a target, which is intended to result in a change of the state (status) of the target. Event can be determined as an occurrence selected from the input message traffic on the basis of a message processing. An event is assigned triple including a subject (an active object, fulfilling action, for example, user or program), an object (a passive object, over which an action is executed, for example, disk, directory, file), an action (for example, reading, recording, execution etc.), and also additional parameters. Examples of events are as follows: an input (output) of the access subjects in (from) a system; start-up (completion) of programs and processes (jobs, tasks); access of the access subjects to defended files, including their creation and deletion; message transmission on data links, etc.

Significant event — an event "significant" from the viewpoint of a security assurance task.

Critical significant event — a significant event that enables a security system to perform a protective activity, because otherwise irreversible dangerous aftereffects could occur. For example, security system can immediately disable all actions supposed by the critical event and disable the connection itself.

Audit records (data) — a fixed (in database, files, etc.) sequence (chain) of significant events.

Pattern (tag, profile, sample) — an informative (from intrusion detection task viewpoint) significant event or an ordered significant event sequence associated with normal or/and abnormal actions. For example, a pattern may be associated with the profile of a user's normal behavior or with the profile of a user's abnormal behavior (the unauthorized access script). Patterns play the roles of basic features, evidences of intrusion attempts that are used by intrusion detectors.

Patterns may be *Simple sequence of significant events* that characterize an attempt to access a specific port of the host or may be *Combined structured sequences of significant events distributed across multiple hosts* within an arbitrary period of time.

Rule — a logic formula specified over a subset of patterns (and/or over their attributes) thus forming higher-level concepts. Rule may be a logic formula specified over such higher-level concepts that belong to different levels of generalization. Examples of rules are *Access control rules* managing access rights of the subjects (users, programs) to objects (files, directories, hosts, etc.). One simpler example of rules is *Identification and authentication rules* that are intended to confirm or reject authenticity of the subject and the identifier submitted. *Intrusion detection rules* may represent knowledge of an intrusion detection agent used for classification of connections in order to distinguish normal, suspicious or abnormal behavior of a subject.

Activity — a concept specified in terms of a single rule or in terms of a conjunction (or disjunction) of rules each given over a set of patterns. Each activity intends to achieve a particular goal. A goal of the activity may be altering the state or status of the respective system objects. In terms of significant events that are arguments of patterns, an activity is a number of (ordered) sequences of significant events and associated actions comprising, for example, data reading, copying, modification, deletion, etc.

Authorized activity — an activity that is consistent with the access control rules.

Non-authorized activity is one that contradicts the access control rules.

Scenario of activity — an ordered or a partially ordered in time set of activities that aims typically to achieve a goal through achieving sub-goals. In this notion, each particular activity aims to achieve a particular goal. A scenario of activity may be distributed over the network entry points and in time. Distributed network-based attack is an example of activities represented in generalized form by a scenario.

Vulnerability — a weakness in a computer system allowing unauthorized action.

Design vulnerability — vulnerability inherent in the design or specification of hardware or software whereby even a perfect implementation will result in vulnerability.

Implementation vulnerability — vulnerability resulting from an error made in the software or hardware implementation of a satisfactory design.

Configuration vulnerability — vulnerability resulting from an error in the configuration of a system, such as having system accounts with default passwords, having "world write" permission for new files, or having vulnerable services enabled.

Connection (session) — a sequence of significant events associated with the single entry of a particular user from entry time till breaking time. The concept of connection is used to detect activities and to associate activity or activities with the particular user. Note that scenario of activity may be detected on the basis of several connections that must be identified as one through processing several connections at an upper level. It is supposed that intrusion detection system possesses the last functionality. Connections may be partitioned into two groups, i.e. a group of actual connections and a group of completed connections. Connections are stored in a database.

Actual connection is specified, as a rule, at two levels. At the first level it is specified as a *Sequence of significant events* and at the second one this specification is generalized and is represented in terms of *Sequence of patterns or activities*.

The model of *Completed connection* is of two-level as well. At the first level it is specified in terms of *Sequence of patterns or activities*. At the second level a completed connection is specified in terms of assigned *Category* that corresponds to the decision made regarding whether it is normal or abnormal and, possibly, is assigned the name of class of abnormality if any.

Attack target — a computer or network *Logical entity* (account, process, or data) or *Physical entity* (component, computer, network) that is the goal of the attack.

Account — a domain of user access on a computer or network that is controlled according to the record of information, which contains the user's account name, password and use restrictions.

Process — a program in runtime consisting of the executable program, the program's data and stack, its program counter, stack pointer and other registers, and all other information needed to execute the program.

Data — representations of facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automatic means. Data can be represented in the form of *Files* in a computer's volatile or non-volatile memory, or in a data storage device, or in the form of *Data in transit* across a transmission medium.

Component — one of the parts that make up a computer or network.

Computer — a device that consists of one or more associated components, including processing units and peripheral units, that is controlled by internally stored programs, and that can perform substantial computations, including numerous arithmetic operations, or logic operations, without human intervention during execution.

Network — an interconnected or interrelated group of host computers, switching elements, and interconnecting branches.

Attacker — an individual who attempts one or more attacks in order to achieve an objective.

Attack tool — some means that can be used to exploit vulnerability in a computer or network. Sometimes a tool is simple, such as a user command, or a physical attack. Other tools can be very sophisticated and elaborate, such as a Trojan horse program, computer virus, or distributed tool.

Physical attack — a means of physically stealing or damaging a computer, network, its components, or its supporting systems (such as air conditioning, electric power, etc.).

Information exchange — a means of obtaining information either from other attackers (such as through an electronic bulletin board), or from the people being attacked (commonly called social engineering).

User command — a means of exploiting vulnerability by entering commands to a process through direct user input at the process interface. An example is entering Unix commands through a Telnet connection, or commands at an SMTP port.

Script or program — a means of exploiting vulnerability by entering commands to a process through the execution of a file of commands (script) or a program at the process interface. Examples are Shell script to exploit a software bug, Trojan horse login program, or Password cracking program.

Autonomous agent — a means of exploiting a vulnerability by using a program, or program fragment, which operates independently from the user. Examples are computer viruses or worms.

Toolkit — a software package that contains scripts, programs, or autonomous agents that exploit vulnerabilities. An example is the widely available toolkit called *rootkit*.

Distributed tool — a tool that can be distributed to multiple hosts, which can then be coordinated to anonymously perform an attack on the target host simultaneously after some time delay.

Data tap — a means of monitoring the electromagnetic radiation emanating from a computer or network using an external device.

With the exception of the physical attack, information exchange and data tap categories; each of the tool categories may contain the other tool categories within them. For example, toolkits contain scripts, programs, and, sometimes, autonomous agents. So when a toolkit is used, the scripts and programs category is also included. User commands also must be used for the initiation of scripts, programs, autonomous agents, toolkits and distributed tools ([Howard-98]).

Attack — a series of steps taken by an attacker to achieve an unauthorized result. An attacker uses some tool to exploit vulnerability in order to cause a needed event. Attack is expressed as an abnormal part of connection, abnormal connection or a number of connections classified together as abnormal. A class of abnormality may be detected or undetected. As a rule, network attack includes phases of Reconnaissance, Implantation and Threat Realization.

Reconnaissance consists in Identification of Hosts, Identification of Services, Identification of OS, Collection of additional information, Resource enumeration, Users and groups enumeration or Applications and banners enumeration.

Identification of Hosts can be realized by *Network ping sweeps* or *Port scanning*.

Port scanning is used for Identification of Services. Port scanning can be fulfilled by different methods: TCP connect scan, TCP SYN scan, TCP FIN scan, TCP Xmas Tree scan, TCP null scan, UDP scan, Half scan, Scanning “FTP bounce”, Dumb host scan, “Proxy”- scanning, etc.

Implantation and threat realization includes, as a rule, the phases of Getting access to resources, Escalating privilege, Gaining additional data, Threat realization, Covering tracks and Creating back doors.

Getting access to resources can be realized by Direct connection to a shared resource, Installation of backdoor server daemons and trojans, Exploitation of known server application vulnerabilities, Cracking of PWL file, Anonymity access to Ftp server, Brute force password guessing, Password stealing attack, etc.

Escalating privilege is accomplished by Password cracking (use of John or L0phtcrack) or Use of known exploits (use of Ls_messages, getadmin or sechole).

Gaining additional data includes Evaluating the trust relations and (or) Searching for cleartext password.

Threat realization can be Confidentiality violation (information disclosure) realization, Integrity violation (information corruption) realization, Availability violation (denial of service) realization, and Theft of resources realization.

Denial of service can be realized through SYN flood, Land, Ping flooding, Smurf, Ping of Death, UDP flooding, storm of inquiries to FTP-server, etc.

Covering tracks includes Clearing logs or (and) Hiding tools.

Creating back doors can be fulfilled by Creating the rogue user accounts, Scheduling batch jobs, Infecting startup files, Planting remote control services, Installing monitoring mechanisms or (and) Replacing apps with trojans.

Unauthorized result of activity (attack) — an unauthorized consequence of events, the logical end of a successful activity (attack). If successful, an activity (attack) will result in one of the following:

Increased access, Information disclosure, Information corruption, Denial of service, Theft of resources.

Increased access — an unauthorized increase in the domain of access on a computer or network.

Information disclosure — dissemination of information to anyone who is not authorized to access that information.

Information corruption — unauthorized alteration (including destruction) of data on a computer or network.

Denial of service — enabling of an intentional degradation or blocking of computer or network resources.

Theft of resources — unauthorized use of computer or network resources.

Attack (intrusion) detection technique — a formal or informal method used for attack (intrusion) detection.

Misuse detection — an intrusion detection technique, which uses patterns of well-known attack or weak spots of the system or some other information and/or technique to identify type of intrusion.

Anomaly detection — an intrusion detection technique that tries to determine whether deviation from established normal usage patterns can be flagged as intrusions.

Intrusion detection agent (IDA) — specialized agent that performs cooperatively with other agents or other software an intrusion detection task. IDA is responsible for detection of attacks, search for unprotected entry points into the host caused by software/hardware "bugs", monitoring of connections and system calls, gathering of pre-processed audit data.

Each of the represented concepts may be considered in more detail at lower level of the ontology.

2.6. Intrusion Detection Learning Application Ontology

The intrusion detection learning application ontology specifies the high-level knowledge regarding fundamental concepts peculiar to the above domain. The structured representation of the basic notions of the Intrusion Detection Learning domain ontology is as follows ([Bass-00], [Cohen-95], [Fayyad *et al*-96], [Forrest *et al*-96], [Kumar *et al*-95], [Lane *et al*-97b], [Lee *et al*-97], [Lee *et al*-98a], [Lee-99], [Mannila *et al*-95], [Porras *et al*-98], [Srikant *et al*-95]):

- Common intrusion detection learning
- Intrusion detection meta-learning
- Local data source intrusion detection learning
- Audit
  - Event registration (audit data gathering)
  - Audit data analysis
- Audit data base
  - Audit log
    - Audit file
      - Audit record
        - Audit token
          - System attribute
  - Audit data
    - Host-based audit data
      - Operating system audit trail
        - System event (calls) sequences (traces)
          - Normal trace
            - Process identifier
            - System call "number"
          - Abnormal traces
            - Exploit identifier
            - System call "numbers"
          - Sliding window
        - Application events sequences

- System log
  - Users' command
    - Subject (user/process identifier)
    - Object
      - Disk
      - Directory
      - File
      - Process
    - Action
      - Read
      - Write
      - Delete
      - Execute
    - Additional parameters
  - Application audit data
- Network-based audit data
  - Network packet attributes and data
    - TCP packet header attribute
      - Time stamp
      - Source *IP*-address
      - Source port
      - Destination *IP*-address
      - Destination port
      - Flags ("*SYN*", "*FIN*", "*PUSH*", "*RST*", or ".")
      - Data sequence length in the packet
      - Data sequence length in the data expected in return
      - Number of bytes of the receive buffer space available
      - Indication of whether or not the data is urgent
    - *UDP*-packet header attribute
      - Time stamp
      - Source *IP*-address
      - Source port
      - Destination *IP*-address
      - Destination port
      - Length of the packet
    - Network packet data
  - Network connections attribute
    - Basic properties of individual *TCP* connection
      - Duration of the connection
      - Type of the protocol
      - Network service on the destination
      - Number of data bytes from source to destination
      - Number of data bytes from destination to source
      - Flags ("*SYN*", "*FIN*", "*PUSH*", "*RST*", or ".")
      - Flag (normal or error status of the connection)
      - Land tag (*1* if connection is from/to the same host/port; *0* otherwise)
      - Number of "wrong" fragments
      - Number of urgent packets
    - Content features within a connection suggested by domain knowledge
      - Number of "hot" indicators
      - Number of failed login attempts
      - Logged tag (*1* if successfully logged in; *0* otherwise)
      - Number of "compromised" conditions
      - Root shell tag (*1* if root shell is obtained; *0* otherwise)
      - *Su* attempted tag (*1* if "*su root*" command attempted; *0* otherwise)
      - Number of "root" accesses

- Number of file creation operations
- Number of shell prompts
- Number of operations on access control files
- Number of outbound commands in an *FTP* session
- Hot login tag (*1* if the login belongs to the "hot" list; *0* otherwise)
- Guest login tag (*1* if the login is a "guest" login; *0* otherwise)

- Traffic feature computed using a time window (in the past *n* seconds)
  - Number of connections to the same host as the current connection in the past *n* seconds
  - % of connections to the same destination host as the current connection in the past *n* seconds that have "*SYN*" errors
  - % of connections to the same destination host as the current connection in the past *n* seconds that have "*REJ*" errors
  - % of connections to the same destination host as the current connection in the past *n* seconds to the same service
  - % of connections to the same destination host as the current connection in the past *n* seconds to different services
  - number of connections to the same service as the current connection in the past *n* seconds
  - % of connections to the same service as the current connection in the past *n* seconds that have "SYN" errors
  - % of connections to the same service as the current connection in the past *n* seconds that have "REJ" errors
  - % of connections to the same service as the current connection in the past *n* seconds to different hosts

- Audit data of other sources

- Algorithms of learning (data mining) for intrusion detection domain
  - Classification
  - Link analysis
  - Sequence analysis (frequent episodes detection)
  - Learned rule sets
    - Learned rule

Let us give definitions (descriptions) of some basic notions of the Intrusion Detection Learning domain ontology. The basic notions are referred to using *italic* font.

Common intrusion detection learning — a generation (modification) of the whole intrusion detection knowledge base.

Intrusion detection meta-learning task — formulations of the intrusion detection learning tasks for the local sources, the intrusion detection knowledge generalization scheme, and the intrusion detection learning of meta-classifiers that uses audit data from one or more sources.

Local data source intrusion detection learning — finding of patterns/regularities/rules based on the local source audit data in accordance with the formulated intrusion detection learning task.

Audit — a process of Event registration and Audit data analysis.

Event registration (audit data gathering) — fixing of information about messages, transmitted to/in the defended network, and (or) security concerned events occurring in a network.

Audit data analysis – a procedure aiming at detecting violations of security policy and at identifying current state of computer system safety.

Audit database — a chronologically ordered log and (or) audit data set. Audit database is the basis for intrusion detection and intrusion detection learning. Audit database can consist of audit logs. *Audit log* includes a sequence of *Audit files,* which are composed of *Audit records.* Each audit record consists of a sequence of *Audit tokens,* which specify *System attributes*. Structures representing audit files have special file tokens in order to mark the beginning and end of file; header and trailer tokens denoting each audit record.

Audit data — records registering occurred events. The audit data fixes results of the system subjects' activity and should be sufficient for restoration, look-up and analysis of a sequence of

operations fulfilled in a defended system. Audit data plays the role of input information for intrusion detection learning and intrusion detection. So two kinds of audit data should be prepared: Training data and Testing data.

The sources for audit records can be such as *Host-based audit data* (Operating system audit trails, System logs and Application audit data), *Network-based audit data,* and (or) *Audit data from other sources*.

Operating system audit trail fixes the *System events (calls)* and *Application events sequences*.

Sequence can be *Normal trace* or *Abnormal trace.* Each sequence (trace) can be represented as a row of the table that has two columns of integers, the first is *Process (Exploit) identifier* and the second is *System call "number"*. These numbers are indices into a lookup table of system call names.

Sliding window is used to scan the traces and create (investigate) a list of (unique) sequences of system calls.

System logs determine different event, including *users' commands*. These event are characterized by *subject* (user/process identifier), *object* (Disk, Directory, File, Process, etc.), *action* (Read, Write, Delete, Execute, etc) and *Additional parameters*.

In *Network-based audit data* for each network event (described by *Network packet attributes and data*) there are attributes defined on the moment of its originating (subject, object, access mode, arrival time), and attributes describing event (message) processing (processor usage time, size of information input in a data link, etc.).

For each *TCP*-packet, the following *TCP-packet header attributes* can be fixed in audit data: Time stamp, Source *IP*-address, Source port, Destination *IP*-address, Destination port, Flags ("*SYN*", "*FIN*", "*PUSH*", "*RST*", or "."), Data sequence number of this packet, Data sequence number of the data expected in return, Number of bytes of receive buffer space available, and Indication of whether or not the data is urgent.

For each UDP packet, the following *UDP packet header attributes* can be fixed: Time stamp, Source IP address, Source port, Destination IP address, Destination port, and Length of the packet.

A lot of different *Network connection attributes* can be defined.

Basic properties of individual TCP connection are Duration of the connection, Type of the protocol, Network service on the destination, Number of data bytes from source to destination, Number of data bytes from destination to source, Flags ("*SYN*", "*FIN*", "*PUSH*", "*RST*", or "."), Flag (normal or error status of the connection), Land tag (*1* if connection is from/to the same host/port; *0* otherwise), Number of "wrong" fragments, Number of urgent packets.

Content features within a connection resulting from domain knowledge are Number of "hot" indicators, Number of failed login attempts, Logged tag (*1* if successfully logged in; *0* otherwise), Number of "compromised" conditions, Root shell tag (*1* if root shell is obtained; *0* otherwise), *Su* attempted tag (*1* if "*su root*" command attempted; *0* otherwise), Number of "root" accesses, Number of file creation operations, Number of shell prompts, Number of operations with access control files, Number of outbound commands in an *FTP* session, Hot login tag (*1* if the login belongs to the "hot" list; *0* otherwise), Guest login tag (*1* if the login is a "guest" login; *0* otherwise).

Traffic features computed using a time window (in the past *n* seconds, for example, $n = 2$) are number of connections to the same host as the current connection in the past *n* seconds, % of connections to the same destination host as the current connection in the past *n* seconds that have "*SYN*" errors, % of connections to the same destination host as the current connection in the past *n* seconds that have "*REJ*" errors, % of connections to the same destination host as the current connection in the past *n* seconds to the same service, % of connections to the same destination host as the current connection in the past *n* seconds to different services, number of connections to the same service as the current connection in the past *n* seconds, % of connections to the same service as the current connection in the past *n* seconds that have "*SYN*" errors, % of connections to the same service as the current connection in the past *n* seconds that have "*REJ*" errors, % of connections to the same service as the current connection in the past *n* seconds to different hosts.

Algorithms of learning (data mining) for intrusion detection domain – algorithms for (automatically) extracting models (knowledge) from large stores of audit data in order to form

knowledge base of intrusion detection system. As a rule, for intrusion detection learning, three *classes* of algorithms are used.

Classification — a class of algorithms that maps a data item into one of classes (normal or abnormal). An ideal application in intrusion detection will be to gather sufficient "normal" and "abnormal" audit data for a user or a program, then apply a learning algorithm to learn a classifier that will determine (future) audit data as belonging to the normal class or the abnormal class.

Link analysis — a class of algorithms that determines relations between fields in the audit database. Finding out correlation in audit data will provide insight for selecting the right set of system features for intrusion detection ([Lee-98]).

Sequence analysis (frequent episodes detection) — a class of algorithms that models sequential patterns. These algorithms can help to understand what (temporal) sequence of audit events is frequently encountered together. These frequent event patterns are important elements of the behavior profile of a user or program ([Lee-98]).

Learned rule sets — a collection of learned rules.

Learned rule (rule) — a logic formula specified over a subset of the audit data patterns and their attributes used for generation (modification) of intrusion detection knowledge base.

Each of the represented concepts considered in more detail at lower levels of the ontology.

2.7. Conclusion

The Chapter presents the developed and implemented software toolkits used in the Project for support of the design, implementation and deployment technology of IDLS. The technology makes use of two software tools. These software tools are Multi-agent System Development Kit (MASDK) and Information Fusion Design Toolkit. The first toolkit, MASDK, mostly supports engineering, implementation and deployment of the reusable components of multi-agent IDLS that weakly depend on the application domain. The second toolkit, Information Fusion Design Toolkit, is responsible for design and implementation of the application-dependent components of multi-agent IDLS.

One of two main components of MASDK is so-called "*generic agent*", while the second one is composed of a number of editors destined for specialization of "generic agent" according to particular application in design. Use of such an approach to multi-agent system design leads to enough flexible technology, in which the target multi-agent IDLS is specified formally in a language (this specification is called MAS "*System kernel*") and afterwards is deployed (installed) within a computer network. In case of necessity of MAS modification the designers can do this through modifying specifications of the respective components of the system represented as *System kernel* and re-generating software agents. At that, such modifications can be only applied to agents' knowledge and data. All architectural solutions incorporated in the reusable agent components remain unchanged.

The main peculiarity of the technology part supported by Information Fusion Design Toolkit is that the latter actually implements a novel kind of IF technology that can be called as "*agent-mediated technology*". This class of technology assumes that design of an IF MAS is performed by distributed collaborating designers, which activity is mediated by a number of agents specifically destined to support for collaboration of designers and dismiss them from a number of routine engineering operations. A number of special protocols described in the Chapter and libraries of external functions make the distributed engineering activity coherent and effective.

The developed multi-level ontology of the intrusion detection learning task unites a structured multitude of basic notions used for specification of the upper levels of the knowledge model manipulated by the components of the system under development. This ontology encompasses the notions from several domain ontologies, i.e. from the "Data fusion" & "Data fusion learning" problem ontology, from the "Intrusion detection" and "Intrusion Detection Learning" application ontologies. The common ontology developed here serves as a basis for design and implementation of the upper-level representation of distributed knowledge that is a shared knowledge for the agents of the multi-agent IDLS. This level of knowledge provides, on the one hand, the integrity of the distributed knowledge base, and on the other hand, the "mutual understanding" of the agents interacting via message exchange.

Chapter 3. Multi-agent Architecture and Operation of Intrusion Detection Learning System

Abstract. The focus of the Chapter is description of the developed multi-agent architecture of IDLS. Architecture is developed on the basis of thorough decomposition of the IDLS functionalities, on analysis of the set of particular system functions and subsequent composition them into groups that are allocated to the particular agents. The basic agent classes are introduced and their collaboration is described. Architecture of the communication platform supporting agents' interaction, classes of communication and interaction protocols used by agents are also briefly outlined.

The Chapter describes also operation of IDS based on information fusion. This description supposes that the system is implemented and deployed in a computer network, learning procedures aiming engineering of the system distributed knowledge base and distributed classifiers have already been successfully completed.

The technology of intrusion detection learning is presented. The Chapter thoroughly describes the learning technology and respective interaction of both IDS and IDLS components. The conceptual explanation of how IDLS operates in classification of the newly received data is given. The accent is on the explanation of the interactions of IDLS agents.

3.1. Architecture of Intrusion Detection Learning System

In this section we present the results of decomposition of the entire task to be solved by multi-agent IDLS, the chosen variant of allocation of the IDLS functionalities to the classes of agents, architecture of IDLS and the agents' communication environment.

One of the objectives of this section is to present exhaustive and structured lists of the tasks allocated to particular agent classes of IDLS and thus to provide the necessary understanding of the roles and responsibilities of agent classes, that in turn must provide understanding of the protocols of agents' interaction. The depth of the entire task decomposition is chosen to the degree that guarantees for the most tasks of the bottom level that they are executed by a particular agent class.

The list of tasks in Tab.3.1 is presented in a semi-structured form. In it, the relation "*Part of*", which is here interpreted as "*Task*"–"*Sub-task*" relation, is represented via the size of indent of paragraphs naming the tasks and subtasks. A general understanding of the senses of the tasks can be concluded from their naming that is made in such a way that, according to our opinion, as much as possible provides self-explanations.

Table 3.1. List of tasks and subtasks of IDLS and their allocation to particular agent classes

Agent's name	Names of tasks and subtasks
KDD Master	Distributed application ontology design Design and providing for consistency of the naming of the set of ontology notions Design of the keys for entity instance identification Application ontology edition Design of meta-model of decision making by IDS as a whole Support for decision making task specification Design of meta-model of IDS decision making Getting and analysis of meta-properties of data of data sources Splitting the data into training and testing samples Design of classification tree Specification of learning tasks Sending of the decision making structure to the particular decision makers of IDS Distributed learning management Meta-classifier learning management Preparation of IDS for use in decision making mode

Agent's name	Names of tasks and subtasks
Meta-level KDD agent	Distributed learning management Computation of (relational) data for training and testing Meta-classifier Meta-classifier learning Mining of rules from meta-data Adjusting the inference mechanism of meta-classifier Testing of the meta-classifier and its performance quality assessment Design of meta-model of decision making by IDS as a whole Sending of the decision making structure to agent-classifier of meta-level
KDD Agent (of a source)	Design of meta-model of decision making by IDS as a whole Receiving of the decision making structure to KDD master Distributed learning management Selection of data for training and testing of Base Classifiers Receiving data form DSM agent Base Classifier learning Mining patterns used by Base Classifier's in decision making procedure Adjusting the inference mechanism of Base Classifiers of source Base Classifiers testing and its performance quality assessment Transformation data represented in numerical scales into discrete ones Sending of the decision making structure to KDD agent of source
Server of learning methods	Learning, training and testing methods management
**[1] Source-based classification agent (base classifier)*	Distributed learning management Classifier learning Receiving of the decision making structure from KDD agent of source Computation of data for training and testing Meta-classifier Base Classifier's decision making Decision making * Base Classifier's decision making Check for incoming of new data Base Classifier's decision making management Firing particular rule of Base Classifier Knowledge Base Making decision by Base Classifier
* *Agent-classifier of meta-level*	Distributed learning management (subject of learning) Meta-classifier learning Receiving of the decision making structure from KDD agent of meta-level Decision making (in IDS operation) Meta-classifier's decision making management Meta-classifier's decision making

[1] Agents marked by symbol "*" are not the subject of the research in this project. These agents are the components of Intrusion Detection System designed by IDLS. Nevertheless, the simplified versions of such agent are also under development due to the necessity to validate the developed technology of IDLS design.

Agent's name	Names of tasks and subtasks
* *Information Fusion (Decision combining) management agent*	Design of meta-model of decision making by IDS as a whole Receiving of the decision making structure Distributed learning management Preparation of the system for operation in decision making mode Decision making Analysis of incoming events participating in classification procedure Decision making management IDS decision making
Data source managing agent	Decision making * Data preparing Retrieval of data samples from data bases of data source[1] Transformation of samples attributes types Computation of training and testing data samples for Classification agents of data source Distributed application ontology design Design and providing for consistency of the naming of the set of ontology notions Design of the keys for entity instance identification Application ontology edition Tuning the gateway for database access Design of meta-model of decision making by IDS as a whole Getting meta-properties of data of data sources Computation of aggregated data properties in reply on a query Getting the list of identifiers of attributes specifying the entity to be classified Extraction from database of the list of identifiers of data to be used for training and testing of IDS Distributed learning management Selection of data for training and testing of Base Classifier Extraction of data from database (for dealing with relational data) Extraction of data from database and its transformation into needed format (for sequences of events) Extraction of data from database and its transformation into needed format (for vector time series of binary data) Computation of data for training and testing Meta-classifier Base Classifier learning Designation the names and types of attributes of data specification that is input data for Base Classifier Meta-classifier learning Designation the names and types of attributes of data specification that is input data for Meta-classifier Data source monitoring to detect receipt of new data*

The interaction of these tasks (and therefore, interaction of the agents responsible for task performance) in the process of IDLS operation specified in terms of protocols is presented in Chapter 2. Interaction of the tasks of applied IDS (respectively, the agents responsible for their performance) designed as the result of the IDLS operation is also considered in Chapter 2.

If the same task is shared by several agents then this means that different agents participate in its solving in distributed manner and these agents are responsible for different subtasks of a shared task.

IDLS can be viewed as the learning component of IDS (Fig.3.1), if it is supposed that the former is from time to time used in off-line mode for incremental learning of IDS.

[1]In our model each data source generates several instances of data of different structures which must by stored in different databases of the same source. Therefore, this agent must provide access to several ODBC sources.

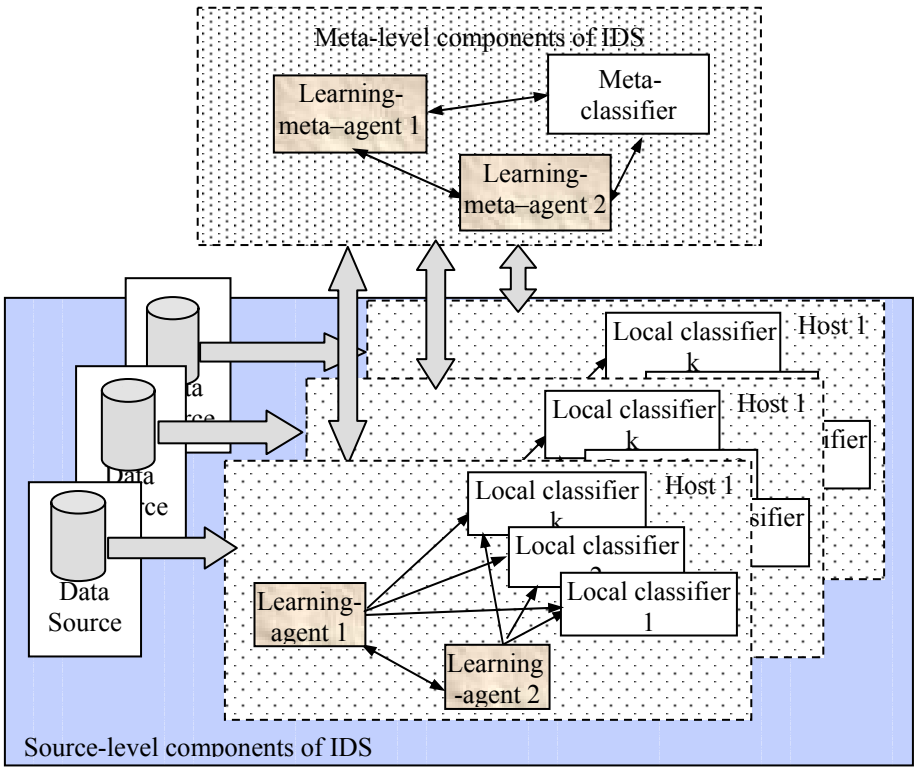


Fig.3.1. General view of common IDS architecture

In our development we use multi-agent architectures for both learning (IDL) and decision making (ID) components. In turn, both of them comprise two types of components. The first type corresponds to those handling with the source-based data; these components are situated in the same hosts as databases of the respective data sources. The second one corresponds to the components manipulating with meta-data generated on the basis of source-based data. The last components can be situated in any host.

In Fig.3.1 the learning components are given in darker color while decision making components are given in white one.

A more detailed architecture of IDL and ID components is depicted in Fig.3.2. In it, the learning (IDL) components (both source- and meta-levels) are depicted in the left hand side and the components corresponding to decision making functionalities (ID) (although both of source- and meta-levels) are depicted in the right hand part of this figure.

Let us outline main components of IDL and ID parts of multi-agent IDLS and their functionalities. Detailed specification of these functionalities including specification of agent interaction protocols corresponding different task is given in the following chapters.

The functions of source-based components of the system under consideration (Fig.3.2, lower part) are as described below.

Data source managing agent

- Participates in the distributed design of the shared components of the application ontology;
- Collaborates with meta-level agents in management of training and testing procedures of particular source-based classifiers and in forming meta-data sample for meta-level training and testing;
- Supports gateway to databases through performing transformation of queries from the language used in ontology into SQL language.

KDD agent of data source

- Trains and tests of source-based classification agents and assesses performance quality of the designed classifiers. In this process, it uses library of training and testing methods, shared and private components of the application ontology and training and testing datasets.

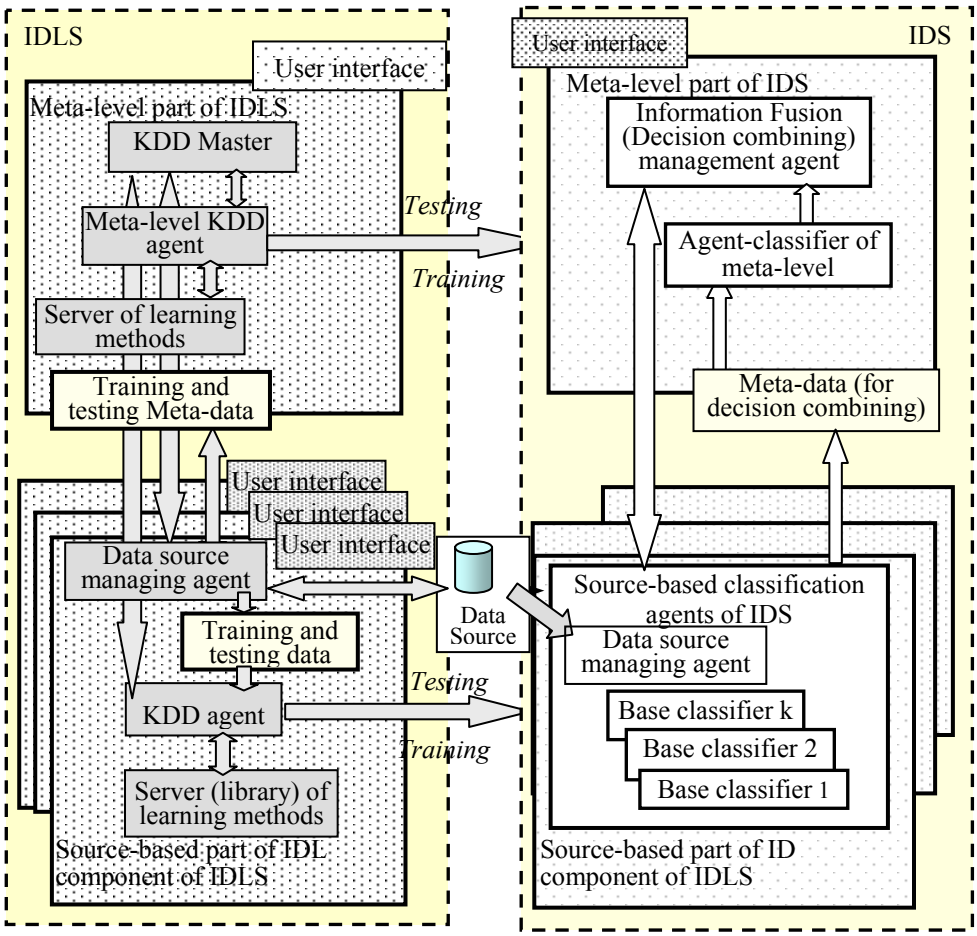


Fig.3.2. Architecture and interaction of IDL (left) and ID (right) components

Classification agent of data source

- Produce decisions using source-based information. It is the subjects of training and testing performed by agents of KDD agent. Classification agent of data source includes several base classifiers (according to the meta-model of source-based decision making).

Server (library) of learning method (not an agent)

- Comprises a set of the software classes implementing particular KDD methods, metrics, etc.

The meta-level components responsible for IDL and ID (Fig.3.2, upper part) and their functions are as follows:

Meta-Learning agent ("KDD Master")

- Manages the distributed design of shared application ontology;
- Computes the training and testing meta-data samples;
- Manages the design of meta-model of decision making.

Meta-level KDD agent

- Trains and tests of meta-level classification agent and assesses its quality.

Decision combining management agent

- Coordinates operation of *Agent-classifier of meta-level* and *Meta-level KDD agent* both in training and decision combining modes of their performance.

Agent-classifier of meta-level

- Produce decisions using meta-level information. It is subject of training and testing performed by *Meta-level KDD agent* of IDLS

*Server (library) of KDD methods (*not an agent)

- Comprises a set of the software classes implementing particular KDD methods, metrics, etc.

The list of functions, libraries or modules developed and their allocation to particular agent classes is presented in Tab.3.2. It should be noted that architecture of IDLS designed by making use of MASDK implements agents behavior (its functionalities) in terms of state machines. Decomposition of the entire IDLS functionalities and composing the subtasks to be allocated to the particular agents is made in implementation-oriented mode. This is the reason why the functions of agents are named in Tab.3.2 as state machines of the respective destinations. Such a terminology is more understandable for designers of IDLS that use MASDK as a design technology support software tool.

Table 3.2. List of functions, libraries or modules and their allocation to particular agent classes

Agent name	Names of function, library or module
KDD Master	*Editor of meta-level ontology*
	State machines providing interaction with editor of meta-level ontology
	Editor of decision making meta-model
	State machines providing interaction with Editor of information fusion meta-model
	State machine querying characteristic of data sources
	Information Fusion meta–model editor
	State machines providing interaction with Information fusion meta–model editor
	State machine responsible for forwarding learning tasks to KDD agents
	State machine responsible for forwarding Decision making meta-model to the decision making agent
	State machine responsible for forwarding training and testing data sample specification
	State machine responsible for preparation of meta-data used for meta-classifier training and testing
	Function querying generalized specification of data sample
	Function querying identifiers
Meta-level KDD agent	*Interface of the meta-learning program tracing (debugger of meta-learning)*
	State machine implementing interaction with meta-learning program tracing
	State machine receiving information about finalizing of the base classifier learning
	State machine receiving meta-learning task specifications
KDD Agent (of a source)	*State machine receiving local learning task specification*
	Basic state machine of user interface supporting training and testing
	Interface of the managers of classifiers' status
	State machine responsible for resending classifiers' attributes
	Interface for estimation of the coverage factor of the rules
	State machine managing rule extraction procedure
	Interface supporting the classifier attribute tuning
	State machine supporting the classifier attribute tuning
	Interface supporting transformation of the data measurement scales
Server of learning methods	*Data mining function "vam"*
	Data mining function "gk2"
	Data mining function "FP-grows"

Agent name	Names of function, library or module
	Data mining function "Temporal mining"
** Source-based classification agent (base classifier)*	*State machine receiving rules generated and classification attributes*
	Decision making state machine of classifier
	State machine informing about readiness of a base classifier to produce decision
	Function responsible for monitoring of arrival of input data
	Decision making based on particular rules
	State machine receiving attributes specifying a classifier
** Agent-classifier of meta-level*	*Decision making state machine of Meta- classifier*
	State machine receiving decision making meta-model
	State machine receiving attributes specifying meta-classifier
Information Fusion (Decision combining) management agent	*Interface of the decision combining support system*
	State machines of the decision combining support system
	Decision combining system
	State machine receiving input data arrived
Data source managing agent	*State machine performing data preparation*
	Function responsible for data extraction and transformation
	State machine receiving specification of notions
	State machine receiving attributes of data
	Interface for tuning of the application ontology notion interpretation
	State machine implementing interface for tuning of the application ontology notion interpretation
	State machine receiving generalized data properties
	State machine performing receiving and forwarding of the identifier's list
	State machine preparing a data sample
	Function responsible for preparing of a data sample
	State machine performing monitoring of the data source
	Function responsible for monitoring of the data source
	State machine responsible for receiving of data attributes
	State machine responsible for receiving attributes of classes

In development of the agents' communication environment the "de facto" standard language is KQML that is used as message content wrapper, whereas the content itself is specified through use of XML language representing message in terms of application ontology. Transport level of message wrapper also corresponds to the standard protocol that is TCP/IP protocol.

The conceptual view of the structure of agent communication within computer network in which multi agent IDLS is situated is depicted in Fig.3.3. It supposes that agent communications are supported by three intermediate components that are:

- Portal of the computer at which an agent sending a message is situated;
- Portal of the computer at which an agent receiving a message is situated;
- Communication meta-agent (agent-facilitator) of the IDLS that provide message addressing.

These components provide the complete transport services for messages.

Protocols needed for support of agent interactions comprises three groups:

1. Protocols that support agent *message exchange* in accordance with the generally accepted (for multi-agent systems) three-level scheme: “message transport protocol” (message envelope) – “message syntax specification” – “message content specification”.
2. Protocols aiming at management of *semantically interconnected dialogs* (conversations) of agents that take place if agents need a cooperation to solve a task. As a rule, such kind of protocols is necessary in case if several agents are involved in a multi-step task solving procedure. This kind of protocols plays the role of meta-level protocols with regard to the protocols of the first group.
3. Protocols supporting cooperative work of agents in distributed design and learning procedures. These protocols are specific for any multi-agent information fusion technology. They are necessary to support for distributed development of data meta-model, ontology, scheme of combining decisions of particular classifiers, and other functions to be performed in order to support for distributed information fusion systems engineering including engineering of IDLS.

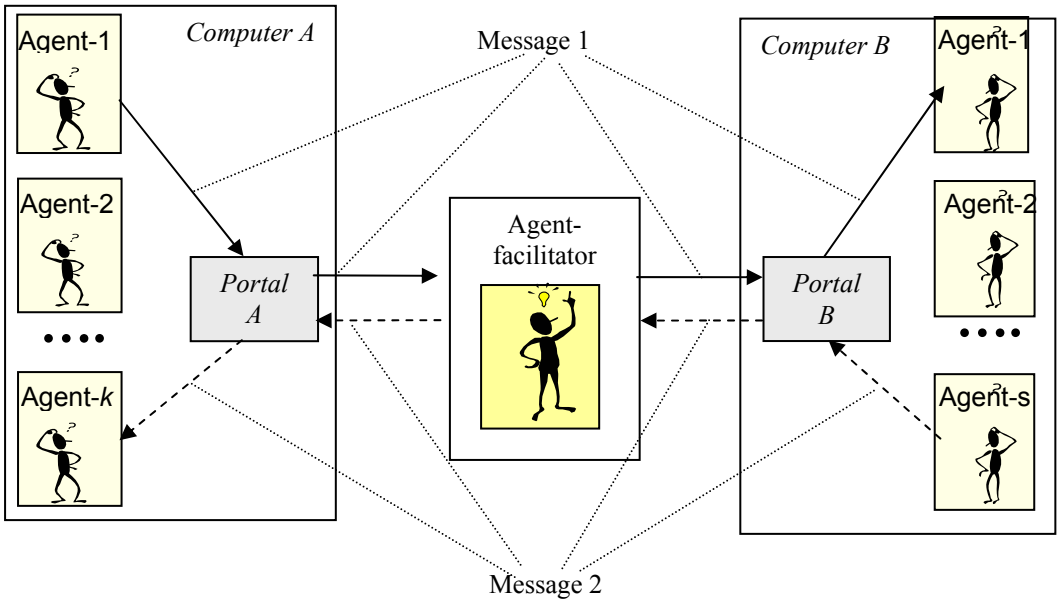


Fig.3.3. Message exchange scheme

This type of protocols plays the role of meta-level protocols in regard to the protocols of the first and the second groups.

Let us conceptually outline the protocols that already implemented.

Protocols of the first group

Transport level of the first group protocols is implemented of the basis of the standard JAVA RMI (Remote Method Invocation) technology that, in turn, is based on usage of standard TCP/IP protocol. The message syntax is represented in KQML language tuned for the purposes of DF applications. Message content (formally, it is not a part of a protocol) is specified in terms of XML language. In the further development of system architecture and technology it is planned to substitute the pure XML by one of its extensions like RDF or DAML+OIL language. Generally say, protocols of this group support coding, transport and agent mutual understanding of each particular message.

The protocols of the second group are realized on the basis of the protocols and respective languages of the first group.

Protocols of the second group

At the current step of the research these protocols are represented as a set of scenarios for processing of a finite set of agent conversations (a conversation is understood as a set of interconnected dialogs of agents solving a shared task). In principle, these scenarios make it possible to support all agent conversations supposed by the IDLS system functionalities. Note that each particular dialog between a pair of agents can activate a series of dialogs of the agent-receiver with other agents if the former needs a help of the latter. In turn, secondary agent-receivers can need a help

of third party agents, and so on. The protocols of this group aim at supporting and management of such multi-step multi-agent dialogs.

It should be noted that these protocols are necessary in every multi-agent system independently of application area. Therefore it is reasonable to develop a reusable software component for such protocol specification, editing and execution. Currently, such a tool is being developed in parallel with the development of particular scenarios for management of agents' conversations. It is a part of the "generic agent" whose conceptual model is considered in detail in Chapter 2. This tool supposes that the whole set of conversation scenarios is represented in terms of a computational model in such a way that each particular scenario corresponds to a "path" within this model. Use of such a tool would provide IDLS system for more flexibility and modifiability of inter-agent conversations.

Protocols of the third group

This group of protocols supports the interaction of agents in process of collaborative design of an applied IDLS system. These protocols are specific for any kind of applications. Protocols of this group are necessary to support for the procedures of distributed development of distributed ontology of an applied IDLS and for management of distributed learning. The protocols of the third group constitute the most important part of the protocol-oriented research within technology for IDLS system design and implementation. Realization of such protocols supposes usage of protocols of the second group that represent the protocols of this group in terms of a number of dialogs. In turn, the latter are implemented in terms of the protocols of the first group.

3.2. Functional Structure and Operation of Generalized IDS

General configuration of generalized IDS based on information fusion (IF) approach and the structure of its distributed knowledge base are depicted in Fig.3.4. In this figure and also in the text below symbol "!" denotes the logical connective "or" while the symbol "/" is below used as separator of alternative (exclusive) events, classes of situation, etc. called further for generality as entities.

According to the architecture of IDS (see Fig.3.2)), it includes the agents of the following classes:

- *Information Fusion (Decision combining) management agent* for brevity called hereinafter *System Managing agent (SM-agent)*. The agent of such a class is always unique in the system.
- *Agent-classifier of meta-level* called hereinafter for brevity Meta-Classifier agents *(MC-agents)*. It can be unique but also can be presented in the system in several instances what depends on the designed meta–model of IDS, that in turn, is determined by the peculiarities of the application;
- Source-based *Base Classifiers* (*BC-agents*) (single agent for each data source);
- *Data source managing agents* (*DSM-agents*) (single agent for each data source).

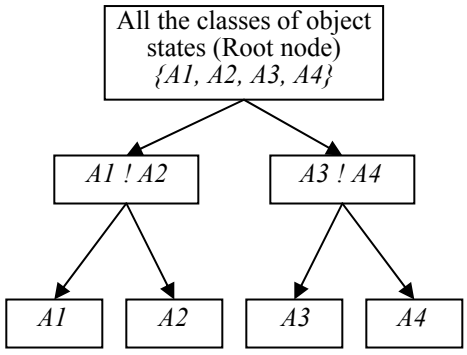


Fig.3.5. Classification tree used in explanations

Below for more clarity of explanations an abstract example of IDS is used. Its characteristics are as follows. The number of data sources is chosen equal to 2. It is supposed that the number of the alternative object states to be discriminated is equal to 4 and the labels of the object states are *A1, A2, A3, A4*. The designed classification tree used in meta-model of Decision Fusion (DF) procedure is presented in Fig.3.5. According to this tree, meta-model of IDS includes 2 meta–classes, *M1=A1!A2* and *M2=A3!A4*. Accordingly, for each of the nodes of this classification tree except its leaves a *meta-model of decision combining* (called also for short *decision tree*) has to be built. Let us remind that each *meta-model of decision combining* (*decision tree*) specifies the structure of decision combining and participating classifiers. Therefore, in the example under consideration IF uses three decision trees at that each of them is mapped to the respective meta-class of the classification tree. The first of the decision trees is destined for discrimination of the meta–

classes *M1* and *M2*, and two others destined for discrimination of the classes *A1* and *A2* of the meta–class *M1* and the classes *A3* and *A4* of the meta–class *M2* respectively (Fig.3.5).

The knowledge concerning IF meta–model that is *classification tree* structure and *decision trees* corresponding to the classification tree *meta–classes*, is a part of SM agent knowledge base (Fig.3.4).

Each *BC-agent* as a rule consists of several base classifiers and *MC-agent* includes all meta-classifiers. The architecture depicted in Fig.3.4 contains singe *MC-agent,* although this is not the necessary case. Respectively, all the three meta-classifiers supposed by IF meta–model are the components of *MC-agent.* Connections *SM- agent* with *MC- agent* reflect the fact that *SM-agent* "knows" the structure of the meta-classification tree and which particular meta-classifier is responsible for combining decisions in each node (*meta–class*) of the classification tree.

Connections between *MC- agent* and *BC- agents* reflect the fact that *MC-agent* "knows" which *BC-agents* and which particular base classifiers of these *BC-agents* are used in the respective tasks corresponding to the *IF meta–model*. Connections between *BC-agents* and *DSM-agents* reflect the fact that the respective *BC-agents* and *DSM-agents* handle with the data of the same source. Connections between *SM- agent* and *DSM- agents* reflect the fact that the former "knows" which data sources are used in IF procedure and which *DSM- agents* are provided with the interface with the respective data sources.

The only agent of the system that is provided with the interface to user is *SM- agent*.

It should be noted that the IDS architecture described in this section is not a general case. Depending on the application, the architectures can be different; in particular, these differences are caused by the followings:

- Several *MC-agents can exist*. The distribution of meta-classifiers among *MC-agents* has only one restriction – all meta-classifiers concerning to one node of classification tree must be the components of the same *MC-agent.*
- Meta–models of combining decisions (decision trees) mapped to different nodes of classification tree can be very different.
- *SM-agent* can be responsible for solving of several IF tasks. In this case each such an IF application can operate according to each particular IF meta–model and different meta–models can differ in both classification trees and the sets of decision trees.

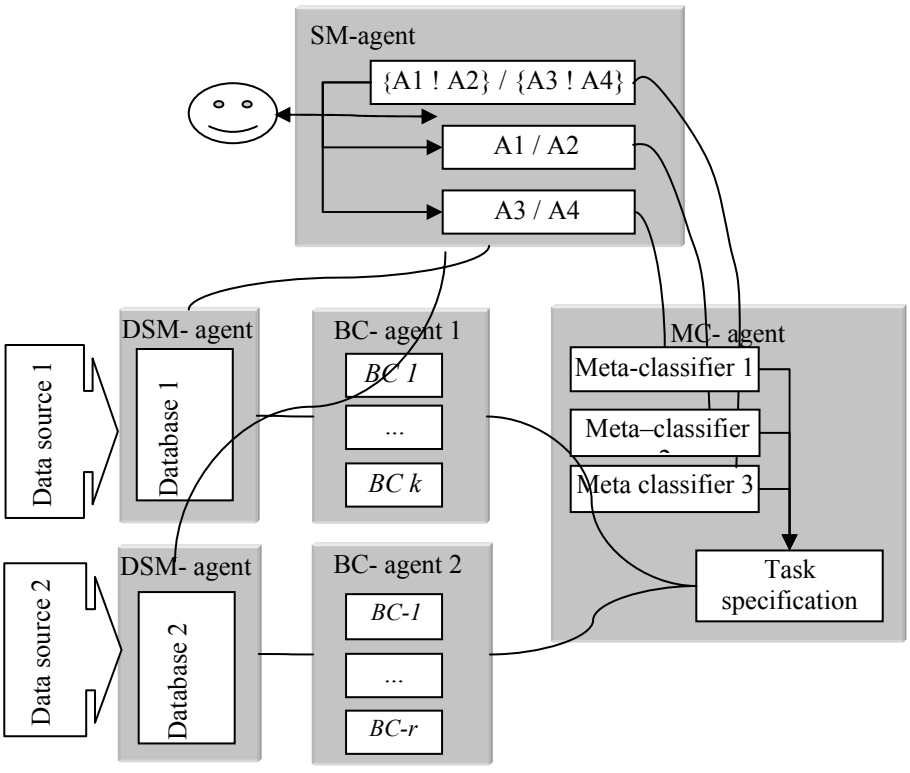


Fig.3.4. A component of the IDS distributed knowledge base

Let us outline the *standard scenario of the IDS operation*.

At the system start, *SM-agent* selects the task corresponding to the root node of the *classification tree*. In our case it corresponds to the task aiming at discrimination of the meta–classes *M1=A1!A2* and *M2= A3!A4*.

Step 1. SM-agent sends the messages to *DSM-* и *MC-agents*. In reply to these messages, *DSM-agents extract the pertinent data from "own" data sources* to be fused. *MC-agent,* in reply to the message received formulates the task corresponding to the root node of the classification tree to be solved by *BC-agents*.

Step 2. In the second step the operation of *BC-agents* is initiated. A signal to start of each *BC-agent* is receiving the respective messages from *DSM-* and *MC-agents*. The messages of *DSM- agents* send to *BC-agents* the vectors of input data specifying an instance of a situation to be classified. Messages from *MC-agent* determines to *BC-agents* specification of the classification sub task to be solved in the current step. While receiving these messages, *BC- agents* initiate the operation of their base classifiers.

Step 3. In this step, *BC-agents* forward to *MC-agent* the solutions produced by their own *base classifiers*.

Step 4. In this step *MC-agent* initiates operation of the meta–classifiers, which combine the decisions received and produce a final result of the first subtask solving. Afterwards *MC-agent* sends the result to the *SM- agent*.

Based on the results of the first subtask, *SM- agent* selects one of the two remaining subtasks and initiates its solving by IF system according to the same scenario. The only peculiarity of this stage is that after the first stage *SM- agent* doesn't send message to *DSM- agents* while in the second stage DSM- *agents* don't send messages to *BC- agents*. The latter is cased by the fact that at the beginning of the previous cycle *DSM- agents* have already computed the input data and sent it to *BC- agents*. Accordingly, in this stage the *BC- agents* have already possessed the input data and that is why are capable to start the task solving.

3.3. Intrusion Detection Learning Scenario

Life cycle of both systems, IDS and IDLS, starts with the development of their configuration (number and nomenclature of agents, as well as their distribution on the hosts of computer network in which the system is going to be deployed). The main information for determining the configuration of both systems is the information about the data source (sources and their location in the system).

The number and location of data sources determines the number and location of *DSM-*, *BC-* and *KDD-agents*. On each host where a data source is located, one instance of each of the above-mentioned agent classes is also located. Regarding agents of other classes, the following rules are applied. Agents of classes *SM* and *KDD master* are always given as single instances. The number of agents of classes *MC-agent* and *Learning agent of Meta–classifiers*, in a general case, depends on the characteristics of the application domain, and there may be several agents of these classes. In the current version of the software implementation an agreement was reached that agents of these classes are also given as single instances.

The generated agents in their original form do not possess any object knowledge. An abstract example of configuration of both systems for two data sources is found in Fig.3.6.

The *process of formation and learning of IDS* includes solving a large number of particular subtasks. A list of the main subtasks may be structured as shown in Fig.3.7. This list partially reflects the order of their execution. Many of the subtasks in that list are quite complex. They, in turn, consist of subtasks, which are also substantial components of the learning technology. A brief description of each of them is given further in this section. More detailed descriptions are to be found in the later subsections.

Design of the shared application ontology. The role of ontology to provide the different agents with a shared set and the same interpretation of notions of the application domain, which is necessary for ensuring a coordinated interaction both between agents and data sources and between agents in the exchange of messages through using the same thesaurus and having the same “understanding” of the meaning of notions in it.

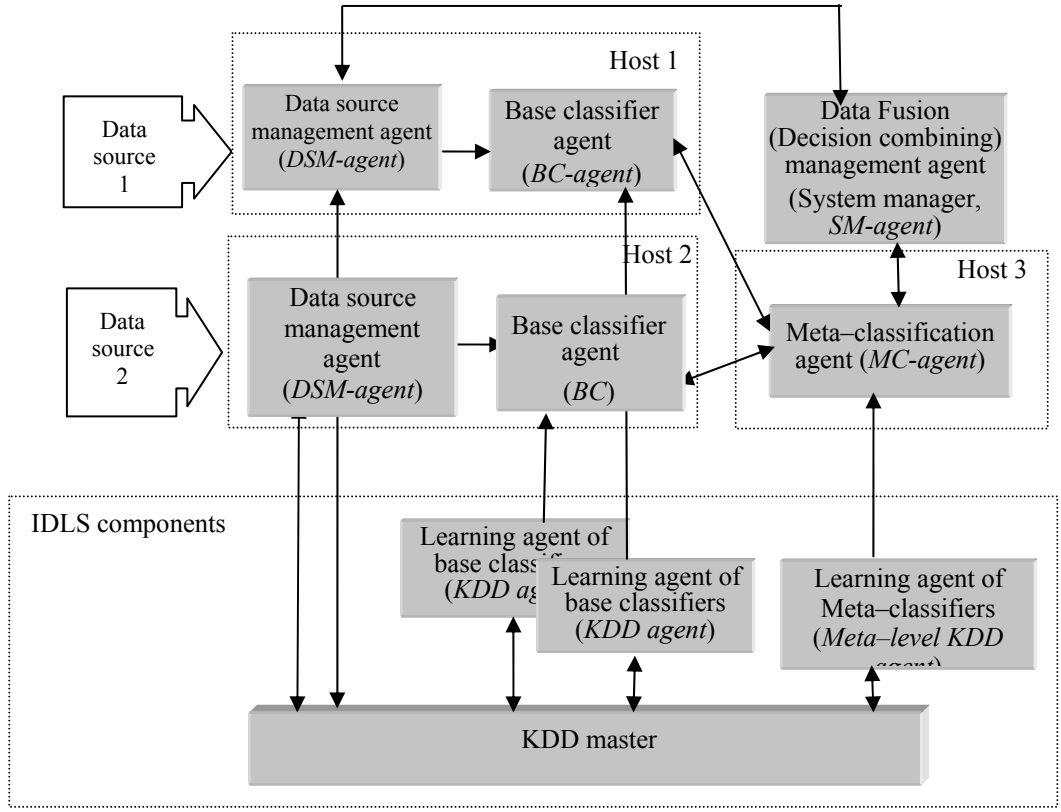


Fig.3.6. Example of initial configurations of IDS and IDLS

Design of binary classification trees. Each *classification tree* corresponds to a separate task solved by IDS. Such tree's leaves correspond to the set of alternative (exclusive) classes of the object's states or situation classes, and its nodes, and to meta-classes, each of which includes several possible states. In a general case, several classification trees with identical sets of leaves may be used for the same classification task.

Design of Meta-model of decision combining (Decision tree). Decision tree is determined for each non-terminal node (meta-class) of *classification tree*. Formation of a *decision tree* implies the description of a meta-model of decision-making at the corresponding node of the classification tree, which describes (1) a structured set of base classifiers and meta-classifiers, and (2) a set of features that are entered into each of the mentioned classifiers.

A description of a block of tasks for the *development of base classifiers' knowledge bases* follows below.

Design of training and testing datasets. In this Project, it is assumed that the expert knowledge is not used in the engineering of the knowledge bases of the IDS components. These knowledge bases are built on the basis of the data mining and knowledge discovery processes. It is assumed that there is an interpreted dataset in each of the sources that is used as the training and testing dataset for the learning of the system's classifiers. An interpreted dataset is a set of instances of specifications of states of objects or situations, to each of which corresponds an identifier (a name) of its class it is belong to. Since IDS uses binary *classification trees*, which, besides the notion "*class*", also utilizes the notion "*meta-class*", it is necessary to solve the task of calculation of the corresponding interpreted sample of training and testing data for the learning of classification in the nodes of the *classification tree* that correspond to meta-classes. The essence of this task consists in choosing the subset of lines of the database in accordance with the condition specified by the description of the meta-class. On the other hand, each base classifier uses a certain subset of characteristics for decision making, therefore the second task that needs to be solved in the computation of the training and testing dataset of base classifiers lies in selecting the columns of the database corresponding to attributes used by base classifier as features for decision making. Other tasks connected to the computation of the training and

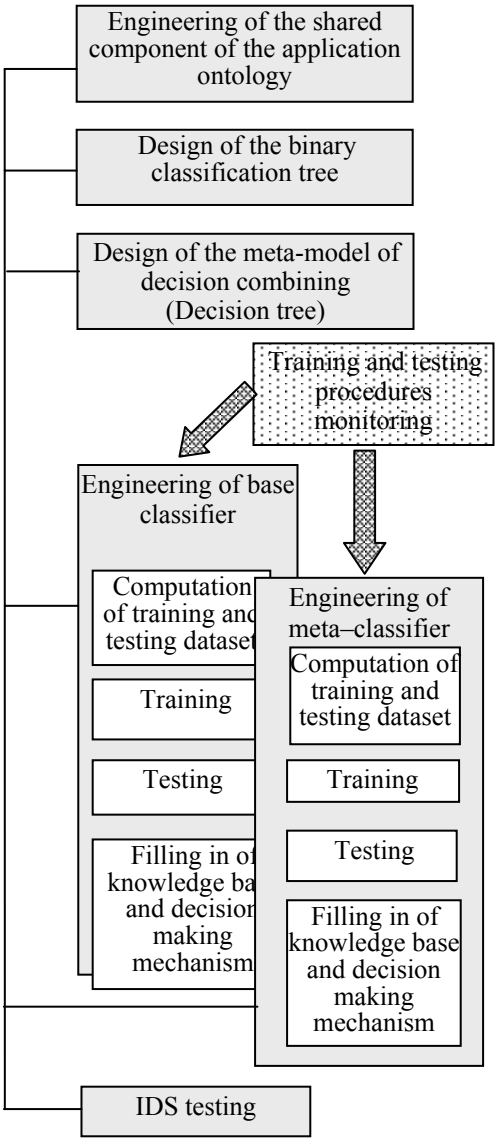


Fig.3.7. Decomposition of the technology of intrusion detection learning

testing data of base classifiers are determined by the necessity to divide instances of databases into those that are used for training and those that are used for testing. Besides, it is also necessary to reserve data for the learning of the meta-classifiers.

Training of the classifier. This task is central to the learning technology suggested. It consists of two main subtasks. The first one consist in building the production rules of the classifier on the basis of the training data through the data mining and knowledge discovery procedures, as well as interactive procedures. The set of production rules is represented through two subsets: a set of arguments voting in favor of one of the classes, and a set of arguments in favor of the other class. The second subtask consists in building a mechanism for processing the arguments and making the decision of the classification task.

Testing of the classifier. Creation of the classifiers' knowledge bases is an iterative process. Each iteration is accompanied by an assessment of the quality of the newly created knowledge base in accordance with the adopted metric based on the data of *confusion matrix*. The quality is assessed based on the testing of the created classifier on a testing dataset. If the assessment results do not comply with the specifications, then the training process must be continued.

Infilling of classifier knowledge base. This procedure is reduced to the overwriting the contents of the classifier's knowledge base obtained at the previous step with the two above-mentioned sets of rules (arguments) and the decision making rule derived at the next step of the learning process.

Below a block of tasks pertaining to the *development of meta-classifiers' knowledge bases* follows.

Computation of training and testing data (meta–data) for meta–classifier learning. By the same labels of classes as in the previous case, the purpose of the task is to form two interpreted subsets of instances specifying the object's states or situations; however, this task is more complex. The input data for the meta-classifiers are not the data specifying the states of object or situations represented in terms of the attributes that are in fact fields of sources' databases. The input data for the meta-classifiers are decisions of base classifiers, i.e. labels of classes to which the base classifiers have referred to specifications of situations, instances of which are represented in the testing sample. Thus, computation of the sample of meta-data used for the learning of the meta-classifier is only possible after the learning procedures for all the base classifiers have been completed. Thus, computation of the training sample for a meta-classifier is reduced to testing the set of base classifiers on the dataset specifically allocated for the training of the meta-classifier.

Training of meta-classifier. The content and sequence of procedures to solve this problem, as well as the form of representation of the results (two sets of arguments "voting in favor" of the corresponding class, and the decision-making mechanism) fully concur with the similar task of training of base classifiers.

Testing of meta-classifier. The content and sequence of procedures to solve this problem fully concur with the ones that have been considered in the task of testing of base classifiers.

Infilling of meta-classifier knowledge base. The content and sequence of procedures to solve this problem fully concur with the ones that have been considered in the task of infilling the base classifier knowledge base.

Testing of IDLS as a whole. Testing of IDLS as a whole after its learning is conducted in the manner analogous to the previously described testing procedures, and can use the same data on which the particular classifiers have been trained and tested, as well as newly-formed datasets and meta-datasets.

Monitoring of the learning process. Fulfilling the tasks in the intrusion detection learning process has a number of characteristic features. Firstly, the tasks have to be solved in a certain order. Secondly, the training of classifiers is an iterative process. Thirdly, the process of design and training of classifiers is distributed in nature; this means that, on the one hand, there is the possibility to solve several tasks in parallel, and on the other hand, there is a protocol for the coordination of distributed learning. The content of the task of monitoring of the learning process is to maintain the order of solving the particular tasks and to maintain coordination of the distributed process.

Several of the key tasks listed here will be further reviewed in more detail.

3.4. Engineering of the Shared Components of the Application Ontology

The design of the shared components of the application ontology, as well as the main data structures used therein, is shown in Fig.3.8. It is assumed that in the initial state, all data sources are defined. The example shows an IDS with two data sources.

The design process of shared ontology can be divided into four stages.

At the *first stage,* the first versions of the data sources ontology (data source local ontology) are designed. It is done by the designers mediated by *DSM-agents*. In a general case, this task is essentially about describing each source's database structures in terms of the language of application's ontology. The final specification of the data source application ontology components is done by a designer mediated by *KDD–master agent*.

At the *second stage,* the designer working through *KDD–master agent* forms the shared application ontology component. Here, the data source application ontology components are used that were created at the first stage. The design of the shared ontology starts with the determining of a list of notions used in intrusion detection task. The representation structure of the notions of the shared ontology includes a key attribute (entity instance's identifier) *ID* and a number of other compulsory attributes. Those attributes are shown in gray in Fig.3.8. They are used to identify instances of the same objects (situations) in the sources' databases (see Chapter 1 in which the conceptual explanation of the *entity instance identification problem* is described in detail). Here, the *ID* parameter is defined as the function of compulsory attributes. For example, the definition of the identifier *ID* of entity *A* uses attributes *Atr 1* and *Atr 2*.

The necessity of using certain attributes in the specification of notions is dictated by the peculiarities of the intrusion detection application. The notions that, according to an expert, may be necessary for the specification of the IDS knowledge base are included in the attributes. Here, since the sources' databases may be different, some of the attributes may have an interpretation in only one of the databases, and others – in several sources' databases (in the example in question – in both databases). The interpretation of an attribute of a notion of the shared ontology is specified in the knowledge bases either explicitly, when they have a field for this attribute, or implicitly, when the interpretation is calculated as an answer to the query to databases as a function of several database fields.

Compulsory attributes used in the calculation of the identifier of the entity's instance must be interpreted in all data sources' databases. Otherwise, the attribute fits as an element of the function that specifies the identifier of the entity's instance.

In the design of the shared ontology, the user working with mediation of *KDD master agent* may specify an arbitrary number of notions. In turn, the notions of the shared ontology correspond to certain notions in the components of local ontology of data sources interpreted in the respective databases. In the database structures, certain relationships may be specified between the notions. Consequently, such relationships must exist between the notions in the shared ontology as well.

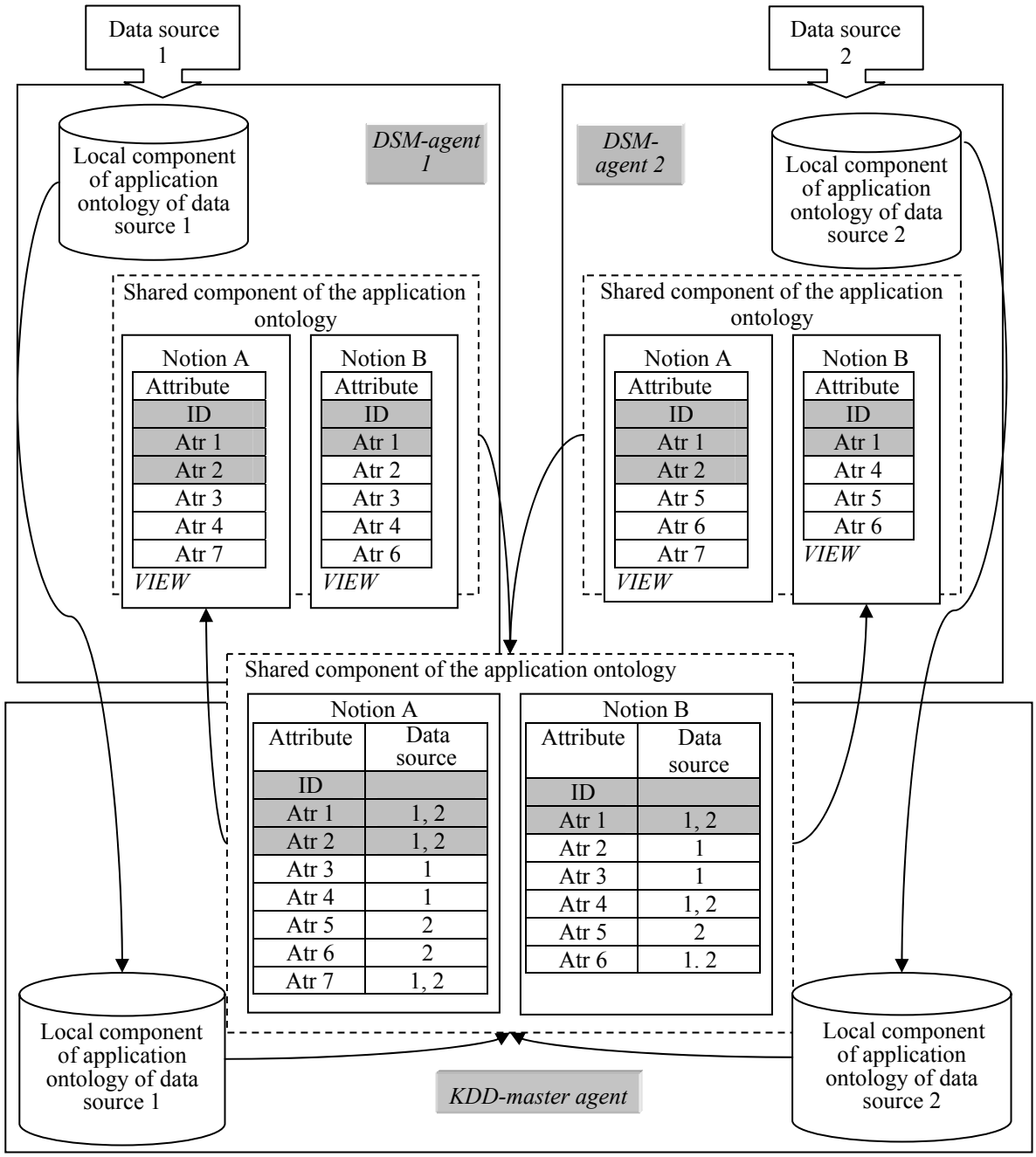


Fig.3.8. Development of the application ontology and ontology structure

After the completion of the second stage of development of the shared component of the application ontology, it is forwarded to *DSM-agents*. Here, each agent is only told the necessary attributes of the notion's specification, namely, names of notions and those attributes that are interpreted in the data source.

At the *third stage,* the final specification of the shared ontology is conducted in distributed mode. Here, *DSM-agents* form interpretations for each notion of the shared ontology in their databases. This task, considering its special character, should be fulfilled by an agent under the supervision of the database administrator.

The *fourth stage* is of formal nature. Here, *KDD-master agent* should get confirmation receipts from *DSM-agents* about the completion of the specification of the shared component of the application ontology. The receipt of such confirmations enables the fulfillment of the next steps of the development of IDLS.

In a general case, shared application ontology should make it possible to solve all the classification tasks of interest. Thus, the next step of the IDLS design is the *specification of these tasks*.

Specification of a single task consists of the following. Firstly, an entity (object) is chosen from the application ontology, whose states' classification is one of the tasks of the system under development. Then, classes of its states are determined. In the simplest case, this task consists in the following. Let us assume that one of the attributes that describe the object's states is supposed to describe the classes of its states. Let us assume this attribute is interpreted in all data sources and is interpreted in the same domain, comprising, for example, the set of states (situations) *{C1, C2, C3, C4}*. Then, the task specification is defined by that set.

It is further assumed that the classification task is determined, i.e. an attribute is specified that describes the required set of classes of states. Let us assume, for example, that the object's states can belong to one of the four alternative classes. This will specify the first task – the development of the IDLS distributed knowledge base. As described earlier in this Chapter, first, one or several binary classification trees are built, determining the upper level of the IF decision making meta–model.

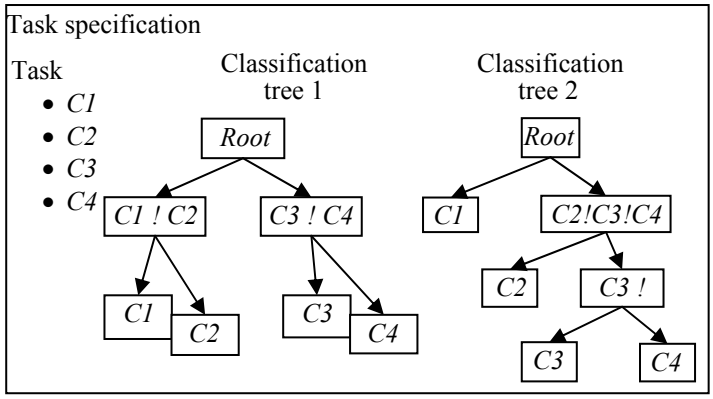


Fig.3.9. Specification of the IDLS task

Classification tree is designed by user mediated by *KDD master agent* in non-distributed way. The result is forwarded to *SM-agent* of IDS and represented formally in the knowledge base of the latter.

3.5. Design of the Structure of Classifiers

Each of the formed binary entity classification trees is used for the structuring of the target system's agents' knowledge bases and for the development of their learning specifications. Here, a number of subtasks are considered separately for each classifier, and other subtasks in the specification process require the joint analysis of the classifiers that have a common relationship to the same binary classification tree (for example, tasks of specifications for classifiers that are related to the same non-terminal node of binary classification tree).

Further in this section, for the sake of definiteness it is assumed that the base that corresponds to the classification task to be solved in node 1 of the tree shown in Fig.3.9 was chosen for creation of fragments of the target system's knowledge bases. This node specifies the target subtask that implies determining one of the two groups of alternatives, *C1 ! C2* or *C3 ! C4*, to which the instance of a situation belongs.

Identifies of base classifier = *BCx*
The task assigned = *(C1 ! C2) / (C3 ! C4)*
# of data source = 1
Training data set – [*IDx ... IDy*]
Testing data set – [*IDz ... IDt*]
Features:
- *A.Atr 1*
- *A.Atr 2*
- *A.Atr 3*
- ...

Fig.3.10. An example of the base classifier specification

Design and specification of the base classifiers

This task, as well as the majority of others, is solved in the context of the shared application ontology. The structure of base classifiers is specified as their unordered list. The composition of data constituting a base classifier specification is presented in Fig.3.10. These data include:

- Base classifier's identifier. It is given randomly, but must be unique.
- Data source used by base classifier.
- Training and testing datasets indicated by the set of identifiers of the interpreted instances of the specifications of their classes.
- Subset of characteristics (attributes) used for training and testing of base classifier. These characteristics are specified in terms of the notions of the ontology. Subset of characteristics

should have interpretations for all instances in the data source, for which the classifier is specified. Besides, it is desirable that the chosen subset of characteristics have the same type of value. This will simplify data mining and knowledge discovery procedures.

Design and specification of the meta-model of decision making and combining

Meta–classifier identifier = *MCx*
IDS task *(C1 ! C2) / (C3 ! C4)*
Training data set – [*IDx ... IDy*]
Testing data set –- [*IDz ... IDt*]
Features (decisions of base classifiers):

- *BC1*
- *BC2*
- *MC1*
- …

Fig.3.11. Example of a meta–classifier specification

Specification of the fragments of state instances in data sources

ID	*ID1*	*ID2*	Comment
Id1			
Id2			+
Id3			
Id4			+
Id5			+
Id6			
Id7			

Fig.3.12. Example explaining how training and testing data sets of meta–classifier are computed

The structure of meta-classifiers is determined for the set of already designed base classifiers. The contents of components that specify a meta–classifier and the data used for its training is shown in Fig.3.11, namely:

- *Meta–classifier's identifier*. It is given randomly, but must be unique.
- *Training and testing datasets*. These are specified by enumerating instances of the object's states represented by their fragments in all sources' databases. Belonging to the same instance is determined through the same value of the key that identifies fragments of the instance's specifications in databases of different sources. The latter is graphically illustrated in Fig.3.12. The figure shows three columns. The first column enumerates all values of instance identifier found in input data. The second and third columns show in gray strings from data sources where this value of the identifier can be found. The figure demonstrates that only three instances (marked by symbol "+") found in the sources' databases describe the same instance; namely, instances with identifiers *Id2, Id4* и *Id5*.

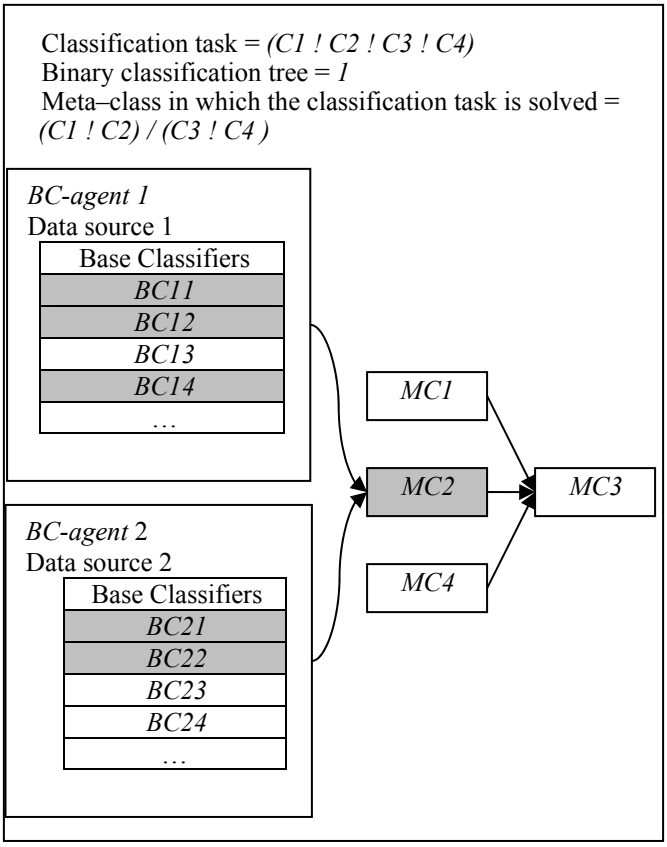


Fig.3.13. Example of the correct specification of the meta-model of decision making and combining

- *Set of characteristics used by meta–classifier in training, testing and in solving data fusion tasks*. These characteristics form a vector, whose components are decisions of base classifiers. If the decision combining structure has several (two or more) levels of meta–classifiers, then the vector of characteristics of the upper meta–classifier is composed of the decisions of meta-classifiers of the lower level. The top level meta–classifier

produces the final decision. In the most cases meta-model of decision combining (decision tree) contains the only meta–classifier.

Representation of the meta-model of decision making and combining in distributed knowledge bases

An example of correct specification of Base classifiers is shown in Fig.3.13. The drawing explicitly shows only connections between meta–classifiers. The explicit illustration of connections between base and meta–classifiers could greatly simplify the perception of this graphic representation. In a general case, each of the base classifiers may form an entry for several meta–classifiers. For the sake of clarity, Fig.3.13 only shows those base classifiers whose decisions constitute a set of input data for the operation of meta–classifier *MC2*.

Meta–model of decision combing is designed by a user mediated by *KDD master agent*. This meta–model representation is stored in knowledge bases of IDS and IDLS. In particular, specification of the base classifiers are stored in knowledge bases of *BC-agents* and also in the knowledge base of *KDD agent* of the data sources.

3.6. Training and Testing of Base Classifiers

This task can be solved after base classifier is provided with specification of the task it is responsible to solve. Training procedure is carried out by the designer who is mediated by the agents situated in the same host as data source. Below the subtask to be solved in training of base classifier are outlined.

Training and testing data of base classifier

The first subtask solved in the process of training of base classifier consists in *choosing data for its training and testing*. In order to choose such data, user sends query containing request for getting such datasets to the *BC-agent*. The query content contains the following data specifying the task of interest: (1) specification training and testing samples; (2) list of attributes of the data source database. These attributes play the role of features used by base classifier for solution of classification task.

The aforementioned query is sent to the *DSM-agent* responsible for its processing. *DSM-agent* processes this query automatically without user's intervention. Results of its operation are table in the database of *KDD agent* of the data source. Examples of tables that contain datasets for training and testing of base classifier are shown in Fig.3.14. The data structure represented there corresponds to the case when the characteristics, in terms of which the base classifier operates, are measured on numerical scales. Fig.3.15 shows the structure of data for a case when characteristics of the base classifier are measured on the binary scale. This structure is of no difference as compared with the structure described earlier, except that the values of characteristics in it are represented in Boolean format.

Preprocessed training data

State		Features				State class
		A.A1	...	*A.Ax*		
Id1		*10*	...	*8*		*C1!C 2*
Id2		*12*	...	*5*		*C1!C 2*
...		...	...	...		...
Id3		*3*	...	*12*		*C3!C4*
Id4		*15*	...	*14*		*C3!C4*
...		...	...	...		...

Fig.3.14.Training data structure prepared for use of VAM algorithm

Preprocessed training data

State		Features				State class
		A.A 1	...	*A.Ax*		
Id1		*t*	...	*f*		*C1!C2*
Id2		*t*	...	*t*		*C1!C2*
...		...	...	...		...
Id3		*f*	...	*t*		*C3!C4*
Id4		*f*	...	*f*		*C3!C4*
...		...	...	...		...

Fig.3.15. Training data structure prepared for use of GK2 algorithm

Base classifier training

Base classifier training includes the following subtasks:

- Engineering (search, mining from training data) of rules (arguments) of classifiers. Note that according to the scheme adopted in the Project, two sets of production rules are formed for each classifier that "argue" in favor each of the two alternative groups of classes of states produced by a certain node (meta–class) of classification tree.
- Rules analysis. The goal of this subtask is to decide which of the rules obtained should be included in the classifier's knowledge base, as well as to decide whether it is advisable to continue searching for new rules.
- Formation of decision making mechanism based on the analysis of the true values of sets of rules (arguments). As a result of this analysis, the classifier under development should make decisions regarding which class one or another instance of the object's state or situation recorded in the data source belongs to.

All the base classifier training subtasks are solved consequently by knowledge engineer mediated by KDD agent of base *classifiers*. The training mechanism automatically chooses the training method that corresponds to the respective structure of training data.

Analysis of the extracted rules to be used by the knowledge base

The process of extracting the set of rules that constitute the classifier's knowledge base is iterative in nature. As a result of each iteration a subset of new rules is formed, which complements the set of rules found at the previous iterations. The analysis of the combined set of rules is aimed at assisting the expert in making the following types of decisions:

- Should the search for new rules be continued?
- Which of the found rules should be included in the classifier's knowledge base?

Note that the rules not included in the classifier's knowledge base are stored in the database of *KDD agent of base classifiers*. This helps avoiding the repetition of the same rule in the further iterations.

Two types of metrics are used in the analysis of the quality of the found rules:

(1) Analysis of the coverage provided by a set of rules. It is said that an instance of training data is covered by a certain rule if its premise and conclusion are true for the respective instance. The formal expression of the coverage analysis of the extracted rules explained in Fig.3.16. The entire set of instances that constitute the training dataset is divided into several (three in this particular case) subsets:

- Subset of instances covered by more than one rule.
- Subset of instances covered by exactly one rule.
- Subset of instances that are not covered by any rules (at least at this point).

This kind of division of subsets is done for the rules and the data that represent both classes of

Analysis of coverage of training data for meta-class *C1 ! C2*

Subset of training data	Degree of coverage	Cardinality of the subset
IDn *= { …, Id, …}*	More than 1	*Kn*
ID1 *= {…, Id, …}*	1	*K1*
ID0 *= {…, Id, …}*	0	*K0*

Analysis of coverage of training data for meta-class *C3 ! C4*

Subset of training data	Degree of coverage	Cardinality of the subset
IDn *= { …, Id, …}*	Более 1	*Kn*
ID1 *= {…, Id, …}*	1	*K1*
ID0 *= {…, Id, …}*	0	*K0*

Fig.3.16. Data structure destined for analysis of the rule coverage

states. The resulting information helps to make conclusions about which of the instances of the training data should be used in the next iterations in the search for rules. Note that the use of the above-mentioned information lies at the basis of the acceleration of the search of rules for the purposes of classification tasks. Such kind of training procedures is typically called "*boosting*" procedures.

(2) Evaluation of the quality of a particular rule. Evaluations of the quality of a particular rule allow for comparing rules and choosing subsets of rules to be included in the classifier's knowledge base. To calculate different evaluations of the quality of production rules, the so called "*confusion matrix*" is used. The example therein uses it for the evaluation of the quality of the rule *"if Exp, then (C1 ! C2)"* (Fig.3.17). Fields of the table shown in gray determine the quantities of specifications in the input data that possess the following characteristics:

- *Na* – the number of examples in training data set, in which *Exp=true* and *C1!C2=true,*
- *Nb* – the number of examples in training data set, in which *Exp=true* and *C3!C4=true,*
- *Nc* – the number of examples in training data set, in which *not Exp=true* and *C1!C2=true,*
- *Nd* – the number of examples in training data set, in which *not Exp=true* and *C3!C4=true,*
- *N(Exp)* the number of examples in training data set, in which *Exp=true,*
- *N(not Exp)* – the number of examples in training data set, in which *not Exp=true,*
- *N(C1!C2)* the number of examples in training data set, in which *C1!C2=true,*
- *N(C3!C4)* – the number of examples in training data set, in which *C3!C4=true,*
- *N* – the size of the training data set.

An example of the evaluation of the quality of this table is an empirical (statistical) evaluation *P* of the correct classification. This evaluation is computed in the following manner: *P = Na / (Na+Nb).*

	C1!C2	*C3!C4*	
Exp	*Na*	*Nb*	*N(Exp)*
not Exp	*Nc*	*Nd*	*N(not Exp)*
	N (C1!C2)	*N (C3!C4)*	***N***

Fig.3.17. Structure of the confusion matrix

Testing of classifier

Testing of classifier uses the same algorithms as the evaluation of the quality of a single base rule. The distinction of classifier testing lies in the fact that in this case, data not used for training, or different from the data used for training, is used as input data (as in, for example, the widely known *cross-validation* strategy). Besides, testing of classifier is aimed in general at quantifying the degree of reliability of its decisions. These evaluations are further used by meta–classifiers in combining decisions made by different base classifiers, since essentially they specify the degree of trust the meta–classifier should have for the decisions made by different base classifiers.

"Filling in" of base classifier

Filling in of Base Classifier consists in recording into its knowledge base the rules (arguments) found for it by *KDD agent of base classifier* and also mechanism of making decisions on the basis of analysis of truth values of these rules. If *BC-agent* already had a knowledge base created at the pervious iteration then it is partially or fully overwritten.

3.7. Engineering and Training of Meta–Classifier

The process of engineering and training of meta–classifiers in general repeats the same processes for the base classifiers; however, they have a number of characteristic distinctions. These distinctions

arise in solving subtasks described further in this section. The other subtasks are solved in the same manner as the ones considered in the previous section.

Meta–classifier training procedures start after the training processes for all classifiers of the lower level, whose decisions are used by the meta–classifier, have been completed. Thus, if several meta–classifiers that combine decisions of base classifiers have been set in correspondence to the node of a classification tree then their training is conducted strictly in the "bottom-up" manner.

Calculation of data sets used for training and testing of a meta–classifier (such data are usually called *meta–data*) involves all classifiers (that are possibly structured into several levels), whose decisions are used by meta–classifier as input data. At that, all such classifiers must already be engineered, and their knowledge bases must be filled in. Fig.3.18 shows the protocol for agents' interaction in the computation of one record of meta-data. The computation of the entire set of both the input and the testing data requires a consecutive implementation of this interaction scheme.

The difference between this scheme and the basic scheme of IDS agents' interaction lies in the fact that here, the role of *SM-agent* is played by the *Meta-level KDD agent.* In accordance with the interaction scheme shown in Fig.3.18, this particular agent initiates the process of meta-data calculation. For this purpose, it sends two first messages to the *DSM-agents*. These messages in particular contain the value of the identifier of the instance of data, for which the record of meta-data

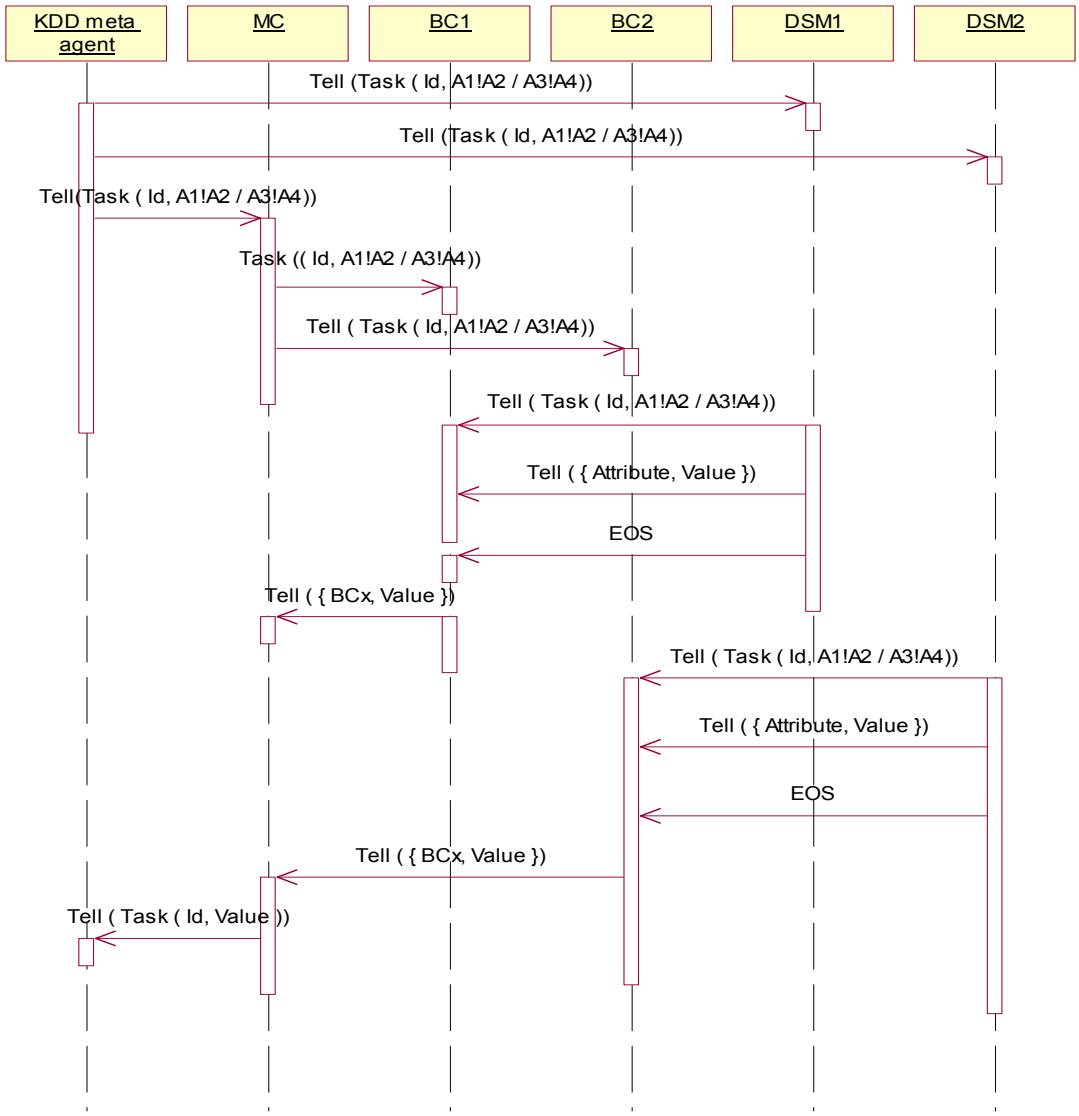


Fig.3.18. Protocol of agent interaction in the process of computation of training and testing meta–data used for meta–classifier learning

must be created in the process of the dialog.

The third message is sent to the *MC-agent* and contains the description of the meta–classifier in training (possibly only partial still). The next fragment of this protocol until the moment when the *MC-agent* receives the decisions made by all lower classifiers is fully identical to the corresponding fragment of the basic interaction protocol for IDS agents. Upon the receipt of these decisions by *MC-agent,* it will record this information into the appropriate meta-database. Note that the partial description of the meta–classifier in the knowledge base of *MC-agent* fully enables its correct operation in implementing this protocol.

The further process of *training and testing of meta–classifier* is fully identical to the process of engineering and training of the base classifier described above.

3.8. Testing of IDS, Monitoring of the Training and Testing Procedures

Testing of IDS after completion of training and testing of all its classifiers (both base and meta–classifiers) can be conducted on both new instances and the instances of the sources' databases. In the latter case, the quality of the formed system can be evaluated. Here, the same methodology for quality assessment can be used that has been described as applied to evaluating the quality of both single rules and classifiers as a whole. This methodology has been reviewed in Section 3.6.

Monitoring of the training and testing processes is aimed at depicting the current state of IDS development and training, as well as controlling the order and execution of particular subtasks.

A user interface shown in Fig.3.19 is used for monitoring. This interface allows for depicting the current state of all classifiers of the meta–model of information fusion. At that, the entire set of classifiers is split into subsets, each of which corresponds to a certain meta–class (node) of *classification tree*. The user interface allows for depicting the state of classifiers that belong to one of its nodes.

Classifiers may be characterized by one of the three states in the process of their development.

- *Specified* – this state implies that the classifier has been specified, and its description has been relayed to *Source-based classification agent,* and the respective base classifier is ready for training.
- *Trained* – this state implies that the trained classifier is represented in the knowledge base of the *MC-agent*.
- *Requires re-training* – this state is analogous to the state "*Trained*" and is only applied to meta–classifiers. It implies that the trained meta–classifier is described in the knowledge base

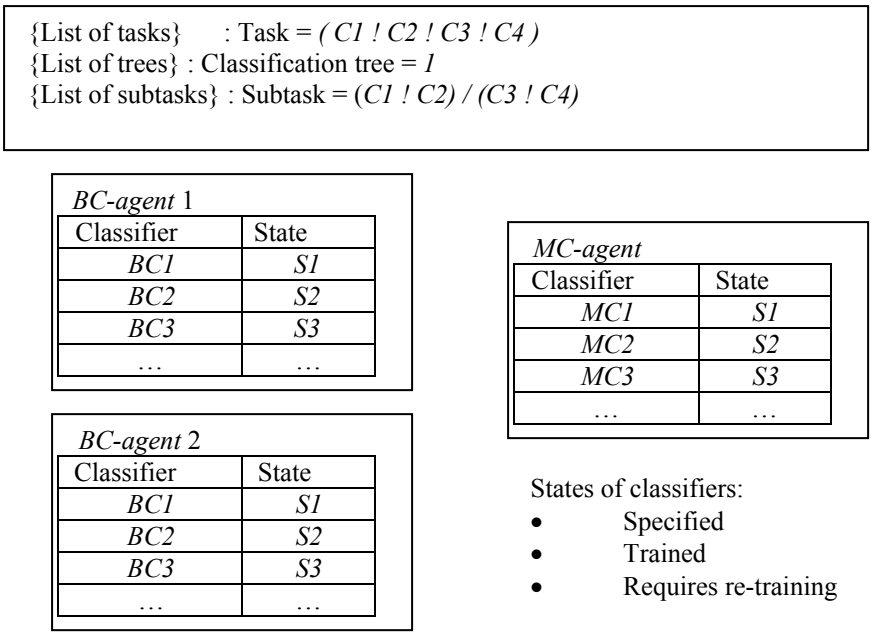


Fig.3.19. Tables of user interface reflecting the state of IDS engineering

of MC-agent, but since one or several classifiers of its lower levels are re-trained, training and testing of this meta–classifier must be repeated.

The final state of training and testing of IDS is such in which all classifiers reached the state "*trained*".

3.9. Conclusion

In this section we presented the results of decomposition of the entire task to be solved by IDLS, the IDLS functionalities allocation to the classes of agents, architecture of IDLS and the agents' communication environment. IDLS is viewed as the learning component of IDS and it is supposed that the former is from time to time used in off-line mode for incremental learning of IDS. We selected several different classes of IDLS agents: KDD Master, Meta-level KDD agent, KDD Agent (of a source), Server of learning methods, Data source managing agent.

In development of the agents' communication environment the "de facto" standard language is KQML that is used as message content wrapper, whereas the content itself is specified through use of XML language representing message in terms of application ontology. Transport level of message wrapper also corresponds to the standard protocol that is TCP/IP protocol. Protocols needed for support of agent interactions comprises three groups: (1) Protocols that support agent message exchange; (2) Protocols aiming at management of semantically interconnected dialogs (conversations) of agents; (3) Protocols supporting cooperative work of agents in distributed design and learning procedures.

The main classes of IDS are Information Fusion (Decision combining) management agent (SM-agent), Agent-classifier of meta-level (MC-agents), Source-based Base Classifiers (BC-agents) and Data source managing agents (DSM-agents). The Chapter showed how IDS operates. It conceptually outlined the operation and interaction of IDS and IDLS agents from functional point of view with the focus on the structure of distributed knowledge bases and on its use in IDS operation.

The process of formation and learning of IDS by IDLS components includes solving a large number of particular subtasks. These are Design of the shared application ontology, Design of binary classification trees, Design of Meta-model of decision combining (Decision tree), development of base and meta–classifiers' knowledge bases, Testing of IDS as a whole and Monitoring of the learning process. A description of each of these tasks was given.