Electronic Notes in Theoretical Computer Science 59 No. 4 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume59.html 15 pages

Rule-based Programming for Building Expert Systems: a Comparison in the Microbiological Data Validation and Surveillance Domain

E. Lamma [a,3] L. Maestrami [5] P. Mello [b,1] F. Riguzzi [a,2]
S. Storari [a,4]

[a] *Department of Engineering,*
University of Ferrara, Via Saragat 1,
44100 Ferrara, Italy

[b] *DEIS,*
University of Bologna, Viale Risorgimento 2,
40136 Bologna, Italy

Abstract

In this work, we compare three rule-based programming tools used for building an expert system for microbiological laboratory data validation and bacteria infections monitoring. The first prototype of the system was implemented in KAPPA-PC. We report on the implementation and performance by comparing KAPPA-PC with two other more recent tools, namely JESS and ILOG JRULES. In order to test each tool we realized three simple test applications capable to perform some tasks that are peculiar of our expert system.

1 Introduction

This work is part of a project for Nosocomial Infection Monitoring. Hospital-acquired or nosocomial infection diseases are developed by patients after the admission to the hospital. They are more dangerous than non-nosocomial infections because they are caused by bacteria that are much more resistant to antibiotics.

The project aims at building an Expert System [1] for Microbiological data validation and bacteria Infections Surveillance (ESMIS). The system validates

[1] Email: pmello@deis.unibo.it
[2] Email: friguzzi@ing.unife.it
[3] Email: elamma@ing.unife.it
[4] Email: sstorari@ing.unife.it
[5] Email: lmaestrami@mail.omnitel.it

the analyses performed by a microbiological laboratory (antibiograms): the quality of analyses results is critical because clinicians use directly these results for therapy definition. ESMIS importa bacteria The system should also identify critical situations such as unexpected resistance of a bacterium or contagion events in an hospital unit and alarm the microbiologist. ESMIS importa bacteria antibiograms, performs validation of the antibiogras controls described in section 3 and returning evaluation results back to laboratory personnel so, in order to perform a significant trial, test application will performs the same steps on bacteria antibiograms.

In the past, DEIS and Dianoema S.p.A. have designed and implemented an expert system for the validation of clinical analyses[2].

In the paper we describe the first ESMIS prototype. It adopts a rule-based approach to identify critical situations and correspondingly generate alarms. The rules applied have been obtained by analyzing international standard guidelines for microbiological laboratory practice and by integrating them with expert suggestions.

The first prototype was implemented using KAPPA-PC [3], a tool for expert system development. A tool has now to be chosen for implementing the final ESMIS prototype. In order to make this choice, we have analyzed the available commercial tools and we have selected the JESS [4] and JRULES [5] for deeper investigation and testing. The feature presented by these tools are described in section 4.

In order to test each tool we realized three simple test applications capable of performing some ESMIS tasks. For each tool, the techniques adopted for implementing these applications and the performance obtained for each trial are described. One criteria for the choice is the performance of the test application because ESMIS will work in real time.

The paper is organized as follows: in section 2 the basic concepts of microbiological data analysis are presented. In section 3 the specifications of ESMIS are dedscribed. In section 4 the various development tools are described. In section 5 the test applications are described together with the performance results obtained. Section 6 presents an overall comaparison of the tools and section 7 concludes the paper.

2 Microbiological Data Analysis

For each isolated bacterium, the antibiogram represents its resistance to a series of antibiotics [6]. The antibiogram is a vector of couples (antibiotic, result), where four types of results are recorded: R if the bacterium is resistant to the antibiotic, I if it is intermediate, S if it is sensitive and null when unknown. Bacteria of the same species with the same antibiograms represent a particular "strain".

Some instruments execute intelligent controls on antibiotic test results but these controls are limited because they don't have information about speci-

men, patient characteristics and infection history. A system, capable of using all available information, may represent a better support for laboratory personnel in the validation task. This system should also control the application of standard antibiotic testing guidelines: these guidelines, used by almost all microbiological laboratories, indicate the execution of antibiotic test methods and the interpretation of results. Examples of problems that this system should solve are: automatic correction of antibiotic results for particular species that present in vitro susceptibility but in vivo resistance, controls on the list of tested antibiotics, predictions of test results for a group of antibiotics using some representative antibiotic (ex. Tetracycline is representative for all tetracyclines), intelligent reporting.

3 Microbiological Surveillance Expert System

A surveillance system may be realized using an Expert System programming approach. This Artificial Intelligence programming technique has been applied to the medical field since 1980. In an Expert System [1], also called Knowledge Based System (KBS), knowledge about the problem is translated into special data structures and rules. An inference engine applies these rules to the available data to perform specific tasks.

Given a newly isolated bacterium and the related antibiogram, ESMIS performs five main tasks: validates the culture results, identifies the most suitable antibiotics list to be reported, issues alarms regarding the newly isolated bacterium, issues alarms regarding patient clinical situation and identifies potential epidemic events inside the hospital. In the validation task, the system finds antibiotics that have not been tested but are necessary, it identifies impossible antibiotic results for a particular species and it tests common relations between antibiotic results. In the intelligent reporting task, the system associates to each antibiotics a suitability, obtained considering some antibiotic characteristics: costs, infection site, bacteria specie and hospital ward. In the bacterium alarms task, the system provides information regarding the bacteria (dangerous resistance, multiresistant bacteria, etc.). In the patient alarms task, it issues alarms regarding the infection history of the patient. Example of possible alarms are:

- Polimicrobic population: if two or more bacteria species where found at successive times in the same sample material;
- Resistance Acquisition: if the newly identified bacterium has more antibiotic resistances than the previous one of the same species.

The system will also provide information regarding the hospital ward (contagion) and epidemic breakout alarms: the system architecture is ready but these controls are not implemented yet.

The knowledge has been elicitated from the annual compendium [8]of NCCLS [7], an international standard organization recognized from almost all

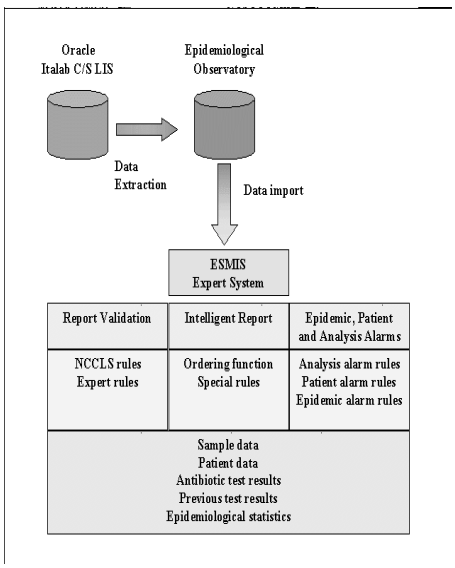

Fig. 1. Databases and data types.

laboratory as reference in routinely work. The compendium contains testing guidelines for microbiological laboratory. NCCLS guidelines, for each species, are basically composed of: a table that specifies antibiotics to be tested, a table that specifies antibiotic test interpretation and a list of exception regarding particular antibiotic test result. We have also used local expert suggestions and particular data mining techniques (see [9]).

A laboratory information system called Italab C/S (developed by Dianoema S.p.A.) manages and stores all the information concerning patients, analysis requests and analysis results in an Oracle database and transfers in real-time microbiological data to a dedicated database called Epidemiological Observatory. A description of databases and data types is shown in figure 1. ESMIS will become part of the routinely laboratory result production process as described in figure 2. ESMIS introduces an automatic validation step in the process: in this step ESMIS presents the results of its controls to laboratory personnel that will decide to agree or disagree with them and to make, if necessary, changes on antibiotic test results. ESMIS also produces the final report and issues alarms regarding the patient clinical situation.

4 Expert system tools

One of the aims of this project is to identify the tool that grants the best cost/benefit ratio for the ESMIS expert system. We decide to test KAPPA-PC, also used in the first prototype implementation, and two other JAVA [10] tools called JESS and JRULES. A tool is basically composed by:

- Libraries of functions for knowledge base construction, rule definition and inference engine personalization;

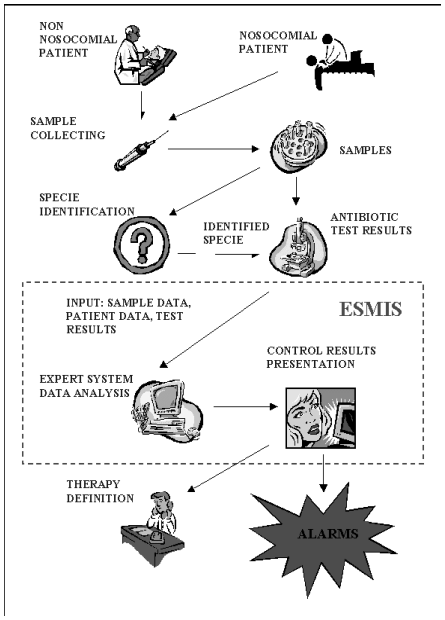

Fig. 2. Control flow of the overall system.

- A graphical developing interface;
- Database connectivity;
- An inference engine;
- A programming language (general purpose, like JAVA and C++, or dedicated).

Each tool has specific characteristics, described in following sections, and may be used for ESMIS implementation in different ways.

4.1 KAPPA-PC

KAPPA-PC is an Intellicorp S.p.A [3] product. Its features are:

- *Programming environment:* KAPPA-PC has a good programming environment composed by some graphical tools for object management, rule management and code debugging. These tools speed up the expert system development and allow easy knowledge management to non-skilled users too.
- *Software architecture:* KAPPA-PC is build with an old programming technology and its 16-bit architecture limits the maximum available memory and therefore the number of objects that can be loaded in memory at the same time.
- *Programming language:* KAPPA-PC uses a dedicated programming language called KAL with a simple C like syntax. This language may be translated in C++ for code compilation.

- *Inference engine:* kappa-PC inference engine may use either forward or backward chaining. Rules are applied using pattern matching in exhaustive mode: in every inference step all rules and all conditions in each rule are tested in order to identify applicable rules. During forward chaining, four different strategy may be used for selecting the rule to be applied from a set of rules with the same priority: bestfirst, depthfirst, breadthfirst or selective.
- *Rule syntax:* Rules are specified with a IF Condition THEN Conclusion syntax. Condition is composed by a set of tests on object attributes linked by logic operator (AND, OR, XOR). Conclusion is composed by a set of actions to be performed on working memory objects.
- *Easy database connectivity:* the tool has a complete function library for database connectivity. Connection is obtained using ODBC drivers.

4.2 JESS

JESS (Java Expert System Shell) [4] is an expert system tool written in JAVA that may be added to every JAVA application. JESS characteristics are:

- *Software architecture:* Since JESS is written in JAVA, applications are independent from the operating system. The code has a 32-bit architecture so there are no memory limitations.
- *Programming interface:* JESS programming interface is represented by a simple console . There are no tools for object management or rule management and the language employs a complex LISP syntax. The development of applications is difficult and not suitable for non-skilled users.
- *Language syntax:* As in the LISP language, the JESS language main concept is that of atom. The head of a list represents a specific function and the elements in the tail represent the function arguments. A variable is an atom in which has ? as first character, and may contain a single atom, a number or a string. Atoms in which the last character is a $ represent multiple variables.
- *Inference engine:* JESS inference engine uses the RETE algorithm. This algorithm grants better performance than standard pattern matching exhaustive techniques. Rules are compiled in a particular net in which each test on an object is executed only one time even if this test is included in more rules. Rules may be applied in forward and backward chaining. In forward chaining two different strategies may be applied for conflict resolution: depth or breadth. In the depth strategy the last activated rule is applied before the other ones. In the breadth strategy rules are applied in the activation order.
- *Rule syntax:* Rules evaluation modifies facts in the working memory. Rules are defined using the "defrule" function using the LISP like syntax. One example is:

```
JESS> (defrule thief_alarm (thief-identified) =>
      (activate-the-bell))
```

This rule is univocally identified by the name "thief_alarm" and subdivided in two parts by "=¿". The first part contains the conditions to be tested and the second the actions to take if all the conditions specified in the first part are verified. The patterns in the conditional part are used for selecting an object of the working memory matching the pattern. In the example shown above, the rule will be activated if the (thief-identified) fact is present in the knowledge base or in the working memory. Each rule has a "salience" that indicates the rule priority.

- Inference on objects: JESS inference engine can not directly interact with java objects so it is necessary to build an interface between JESS objects and JAVA objects. This characteristic makes application development quite complex.

4.3 JRULES

ILOG JRULES 3.0 [5] is a library of JAVA classes used for adding intelligent reasoning to a JAVA application. JRULES has the following features:

- *Software architecture:* JRULES is written in JAVA, so applications are independent of the operating system. The code has a 32-bit architecture so there are not memory limitations.
- *Programming interface:* JRULES has a set of graphical tools for the management of classes, for rule editing and for code debugging. There is also a graphical tool for managing the transfer of data between JAVA classes and database tables.
- *Programming language:* JRULES does not have a dedicated programming language but directly uses JAVA. The inference process can directly manipulate JAVA objects.
- *Inference engine:* JRULES inference engine uses the RETE algorithm. The inference engine works in forward chaining, applying instantiated rules on working memory objects. Rules are selected considering priority and particular criteria (Refraction, Priority, Recency, Lexicographic order of rule names) are used to resolve conflicts when several rule instances are candidates for firing at the same time.
- *Rule syntax:* The conditional part of a rule is composed by one or more expressions (patterns) capable of selecting some objects of the working memory.

```
rule rule_1 {
  when {
   Class1( colour == white; shape == square );
   Class1( shape == circle);   }
```

```
then {...}
}
```

Rule activation may lead to changes in the working memory and rule applicability is tested until new objects are added into it. Rules can be grouped together as rule packets. Associating rules in a packet provides a way to activate and deactivate the rules by using the packet name.

- *Truth Maintenance System:* A Truth Maintenance System has the aim of eliminating facts deduced from hypotheses that have since turned out to be false. It is implemented by means of logical objects. Labeling an object as logical means that its existence is based upon some kind of justification. A logical object is maintained in the working memory as long as the condition part, which constitutes its justifications, remains true. If the condition part becomes false, the logical object loses one of its justifications. A justification counter is associated with the logical object to record the number of times it has been justified. The logical object itself continues to exist in the working memory as long as the value of the counter has not dropped to zero.
- *Database connection:* The JRULES database connectivity tool is a Java application-programming interface (API), which provides access to relational databases. Using the database connectivity tool (DBTool) the user can access any JDBC-compliant database from an ILOG JRULES application. This allows the user to use rules on remote objects and provides a way to construct three-tier architecture applications. DBTool is provided with a graphical user interface (GUI), which allows you to easily select and implement a mapping from database entities to Java objects.

5 Tool testing phase

In order to test the tools we realized three simple test applications, one for each tool, capable to perform the same ESMIS tasks. Each test application is basically composed by:

- *Working memory:* objects containing antibiotic data, patient data and antibiotic test result data;
- *Rule base:* restricted to some validation rules;
- *Application functions:* used for importing data from a database, returning evaluation results to database, managing knowledge and performing actions and controls during the inference process.

In this section we describe, for each tool, the techniques adopted for implementing these components and the performance obtained for each trial.

5.1 Working memory

JESS and JRULES testing environments is based on a set of classes, used for several tasks. The applications import analysis requests and antibiotic

test results using a JDBC interface and create the relative objects. JESS inference engine can not directly interact with java objects so it is necessary to build an interface between JESS objects and JAVA objects. The KAPPA-PC testing application executes the same tasks using an ODBC connections and the database connectivity functions provided by KAL language. Descriptions of the working memory classes and rule base implementation are presented below for each tool.

Working memory classes Each antibiotic test result is an object of a class called 'AntibioticRes' which contains:

- Antibiotic name;
- Antibiotic test result;
- Antibiotic test result proposed by expert system;
- Rack test alarms and justifications;
- Validation alarms and justifications;
- Reporting alarms and justifications;
- ToReport Boolean field, that indicates if test result should be presented in the final report;
- DerivedResult, that indicates if the test result was returned by instruments or derived by other results during the inference process.

Some methods were defined for the 'AntibioticRes' class in order to change the proposed results and to insert alarm justifications. Other implemented classes are:

- A 'Patient' class, containing information about the patient;
- A 'Request' class, containing information about the request;
- A 'IdentifiedMicro' class, containing information about identified bacteria;
- A 'AntibioticHierarchy' class, containing antibiotic hierarchy and characteristics.

The 'AntibiticcRes' class definition used in JESS and JRULES was created using the following JAVA code:

```
public class AntibioticRes {
        public String Code;
        public String Name;
        public String TestResult;
        public String ProposedResult;
        public String[] RackTestComment;
        public String[] ValidationComment;
        public String[] ReportingComment;
        public boolean ToReport = TRUE;
        public boolean Derived = FALSE;
```

```
}
```

'AntibioticRes' class in KAPPA-Pc was created using the graphical class editor.

KAPPA-PC testing application is written using only the dedicated KAL language. JESS and JRULES testing applications were merged into a single JAVA application in which we included the system classes necessary for embedding JESS and JRULES reasoning. In this JAVA application we created an 'ExpertSystem' abstract class containing only abstract methods, that represents the interface for JESS and JRULES subclasses containing method implementations. These classes are instantiated according to user choice:

- *'JrulesExpertSystem' subclass of 'ExpertSystem' class:* In the initialization phase an object of the JRULES 'IlrContextIlr' class and of the JRULES 'IlrRuleset' class are created. The 'IlrContextIlr' object contains the expert system functions and structure, while the JRULES 'IlrRuleset' object contains the application rules. Methods for knowledge base construction (data about bacteria and antibiotics) and inference management are defined too.
- *'JESSExpertSystem' class of 'ExpertSystem' class:* In the initialization phase an object of the JESS 'Rete' class, containing the expert system function, structure and rule format, is created: as previously described, the application can not directly work on JAVA objects, so their JESS description is created specifying each command and executing it by Rete.executeCommand ("JESSCode").

5.2 Rule base implementation

In our testing environment, we have built three lists of rules, containing respectively rules in JESS syntax, rules in JRULES syntax and rules in KAPPA-PC syntax. We formalized 34 rules obtained by analysing the NCCLS document for the Staphilococcus Aureus species, and considering expert suggestions. These rules control antibiotic the validity of test results by working on objects describing antibiotic characteristics and representing relations that need to be verified. A rule example is:

```
Rule_1: IF      TestResult(oxacillin)=R
```

THEN TestResult(PENICILLIN)=R;

This rule may be implemented in different ways depending on used tool.

JESS syntax

```
(defrule rule_1 (declare (salience 2))
        (resantib (CODE OXA) (RIS_PROP R))
        ?r <-(resantib (CODE ?cod) (RIS_PROP ?rp))
                (test (neq ?rp R))
        (antibiotic (CODE ?cod) (FAMILY PENICILLIN))
```

```
        =>
        (bind ?c (fact-slot-value ?r COMMENT))
        (bind ?temp (fact-slot-value ?c COM_VAL))
(modify ?c (COM_VAL ?temp "IF OXACILLIN test result is R THEN
            test results of antibiotics belonging into
            PENICILLIN family must be R."))
        (modify ?r (RIS_PROP R))
)
```

JESS syntax is similar to LISP syntax. In order to make the rule simpler and clearer we have defined some functions for performing common operations. For example, the function 'ValidAlarm' is defined as follows:

```
(deffunction ValidAlarm (?res_ant ?risp ?comment)
        (bind ?c (fact-slot-value ?res_ant COMMENT))
        (bind ?temp (fact-slot-value ?c COM_VAL))
        (modify ?c (COM_VAL ?temp ?comment))
        (modify ?res_ant (RIS_PROP ?risp)) )
```

This function may be included in the conclusion part of a rule:

```
=> (ValidAlarm ?r R " IF OXACILLIN test result "))
```

JRULES syntax

```
rule rule_1
{
packet = Vali; priority = 2;
when
{
 ResAntib(Code.equals("OXA"); PropRes.equals("R"));
 ?r_1: ResAntib(?c_1: CODE; !PropRes.equals("R"));
Antib(Code.equals(?c_1);Family.equals("Penicillin"));
}
then
{
   ?r_1.ValidAlarm("R","IF OXACILLIN test result is R THEN
        test results of antibiotics belonging into PENICILLIN
        family must be R.");
 update ?r_1;
} };
```

This rule verifies three conditions:

- If there exists an object of the 'AntibioticRes' class with the Code field equal to OXA and the test result field equal to R
- If there exists an object R2 of the 'AntibioticRes' class with the test result field not equal to R

- If the antibiotic represented by R2 belongs to the 'Penicillin' group
- If all these conditions are satisfied, then the "ValidAlarm" function, a java method of the 'AntibioticRes' class, is called: this function sets the 'ProposedResult' field to the value of the first function argument and adds to the comment field the value of the second function argument.

KAPPA-PC syntax

Rule_1 con be expressed in KAPPA-PC as follows:

```
If VerifyAntibioticResult(OXA,R) And
    Not(VerifyFamilyResults(Betalammin_1,R));
Then {
    ValidationRuleName(rule_1);
    ValidAlarm:"If OXACILLIN test result is R Then test results
    of antibiotics belonging into PENICILLIN
    family must be R.");
```

In this rule, we used some functions:

- 'VerifyAntibioticResult' verifies if 'Oxacillin' is an element of the patient antibiogram and if its test result is R (resistant);
- 'VerifyFamiltyResults' verifies if there is one or more antibiotics of the antibiogram belonging to the 'Penicillin' family (represented by BetaLattamin_1) whose result is R;
- 'ValidAlarm' issues an alarm for each antibiotic that verifies the rule conditions setting a customisable comment.

5.3 Tool performance

One criteria for final tool choice is represented by the performance of the test applications. ESMIS will work in real time importing bacteria antibiograms, performing the controls described in section 3, and returning evaluation results back to the laboratory personnel. Therefore, in order to perform a significant trial, the test applications will performs the same steps on bacteria antibiograms.

In order to evaluate the test applications reaction to stress situation, we tested them on blocks of increasing dimensions. The evaluation steps are:

- The application retrieves a block of N antibiograms and, for each one, the associated antibiotic test results are imported in 'AntibioticRes' objects and added to working memory;
- 34 validation rules are applied on working memory elements using forward chaining;
- Changes to antibiotic test results (new proposed results and validations note) are applied to corresponding database records.

Table 1 shows the performance obtained using the three different proto-

Table 1
Performance on antibiogram block validation

Antibiogram Number	JRULES Sec.	JESS Sec.	K-PC Sec.	Issued alarms Number	Corrections Number
100	7	7	17	35	35
200	14	15	35	72	72
300	22	23	55	109	109
400	29	29	72	143	143
1000	73	76	190	332	332

Table 2
Comparison between analyzed tools L = Low, M = Medium, H = High P = Poor, S = Sufficient, A = Acceptable, G = Good, VG = Very good

Tool	Language Complexity	Developing Tools	Programming Complexity	Inference	Engine Performance	Cost
K-PC	L	VG	L	A	S	L
JESS	H	P	H	A	G	M
JRULES	M	G	M	VG	G	H

types for the validation of a set of N antibiograms. The sonclusions about results of this first trial are:

- Each prototype correctly applies rules making the same number of corrections on the proposed results and issuing the same alarms;
- Prototypes written using JESS and JRULES are more efficient than KAPPA-PC one because they use a more complex and advanced inference engine (RETE algorithm);
- Prototype performances are not influenced by antibiograms block dimension: relation between elapsed time and the number of antibiogram is linear.

Test was performed in on a PentiumIII 733MHz Intel pc with 128 MB of RAM with Microsoft WindowsNT4.

6 Results of tool analysis

Table 2 synthesizes quality of examined tool features, described in section 4, and their suitability in realizing a system for real-time microbiological data validation and surveillance, described in section 5. This table reports a quality value for: language complexity, developing tools, programming complexity, efficiency and amount of features available within the inference engine, performance, and cost.

JRULES offers the best technical features but because of its high cost it is more suitable for big industrial applications. JESS is powerful but its complex language and poor developing environment make application development slow and limited only to skilled users. KAPPA-PC is a good tool with a simple and easy developing interface and programming language. Its inference engine has low performance and makes it not suitable for high complexity reasoning and time-critical applications.

7 Conclusions

In this paper we described a project whose aim is to build a system for microbiological laboratory data validation and bacteria infections monitoring. We report on the first results we have obtained with a prototype that adopts a knowledge-base approach to identify critical situations and to correspondingly issue alarms. In order to identify the most suitable tool for the ESMIS final prototype implementation, we analyzed three commercial tools for expert system development. We have presented tool specific characteristics and we have described tests executed on simple application realized with such tools.

8 Acknowledgements

We are grateful to A.Nanetti, belonging to Medicine Department of the University of Bologna and to MURST [11](project 23204/DSPAR/99) that has partially supported this project.

References

[1] M. Stefik,*Introduction to knowledge systems*, Morgan Kaufmann, 1995

[2] M.Boari, E.Lamma, P.Mello, S.Storari, S.Monesi, *An Expert System Approach for Clinical Analysis Result Validation*, ICAI 2000,Las Vegas, Nevada, USA

[3] Intellicorp, Inc., `www.intellicorp.com`, Accessed 07/08/2001

[4] JESS, `herzberg.ca.sandia.gov/jess/`, Accessed 07/08/2001

[5] JRULES, `www.ilog.com`, Accessed 07/08/2001

[6] A.Balows, W.J.Hauser,Jr, K.L.Herrmann, H.D.Isenberg, H.J.Shadomy, *Manual of Clinical Microbiology*, 5Ed, American Society for Microbiology, Washington, 1991

[7] National Committee for Clinical Laboratory Standards (NCCLS), `www.nccls.org`, Accessed 07/08/2001

[8] NCCLS, *Performance Standards for Antimicrobial Susceptibility Testing; Ninth Informational Supplement*, M100-S9 Vol. 19 No. 1, January 1999

[9] E.Lamma, M.Manservigi, P.Mello, A.Nanetti, F.Riguzzi, S.Storari, *The Automatic Discovery of Alarm Rules for the Validation of Microbiological Data*, IDAMAP2001, London, UK

[10] JAVA by SUN Microsystems, `www.sun.com`, Accessed 07/08/2001

[11] Ministero dell'universit e della ricerca scientifica e tecnologica, `www.murst.it`, Accessed 31/03/2001