A Knowledge-based Web Information System for the Fusion of Distributed Classifiers

Grigorios Tsoumakas, Nick Bassiliades and Ioannis Vlahavas
Department of Informatics
Aristotle University of Thessaloniki
Greece

ABSTRACT

This chapter presents the design and development of WebDisC, a knowledge-based Web information system for the fusion of classifiers induced at geographically distributed databases. The main features of our system are: i) a declarative rule language for classifier selection that allows the combination of syntactically heterogeneous distributed classifiers, ii) a variety of standard methods for fusing the output of distributed classifiers, iii) a new approach for clustering classifiers in order to deal with the semantic heterogeneity of distributed classifiers, detect their interesting similarities and differences and enhance their fusion and iv) an architecture based on the Web services paradigm that utilizes the open and scalable standards of XML and SOAP.

INTRODUCTION

Recently the enormous technological progress on acquiring and storing data in digital format has led to the accumulation of significant amounts of personal, business and scientific data. Advances in network technologies and the Internet have led to the availability of much of these data online. Personal text, image, audio and video files are today accessible through Web pages, peer-to-peer systems and ftp archives. Businesses have transferred their enterprise systems online providing their customers information and support of excellent quality in a low cost manner. Huge scientific data from physics experiments, astronomical instruments and DNA research are being stored today in server farms and data grids, while at the same time software technology for their online access and integration is being developed. Today, the grand challenge of Machine Learning, Knowledge Discovery and Data Mining scientists is to analyze this distributed information avalanche in order to extract useful knowledge.

An important problem towards this challenge is that it is often unrealistic to collect geographically distributed data for centralized processing. The necessary central storage capacity might not be affordable, or the necessary bandwidth to efficiently transmit the data to a single place might not be available. In addition, there are privacy issues preventing sensitive data (e.g. medical, financial) from being transferred from their storage site.

Another important issue is the syntactic and semantic heterogeneity of data belonging to different information systems. The schemas of distributed databases might differ, making the fusion of distributed models a complex task. Even in the case where the schemas match, semantic differences must also be considered. Real-world, inherently distributed data have an intrinsic data skewness property. For example, data related to a disease from hospitals around the world might have varying distributions due to different nutrition habits, climate and quality of life. The same is true for buying patterns identified in supermarkets at different regions of a country.

Finally, systems that learn and combine knowledge from distributed data must be developed using open and extensible technology standards. They must be able to communicate with clients developed in any programming language and platform. Inter-operability and extensibility are of primal importance for the development of scalable software systems for distributed learning.

The main objective of this chapter is the design and development of WebDisC, a knowledge-based Web information system for the fusion of classifiers induced at geographically distributed databases. It's main features are: i) a declarative rule language for classifier selection that allows the combination of syntactically heterogeneous distributed classifiers, ii) a variety of standard methods for fusing the output of distributed classifiers, iii) a new approach for clustering classifiers in order to deal with the semantic heterogeneity of distributed classifiers, detect their interesting similarities and differences and enhance their fusion and iv) an architecture based on the Web services paradigm that utilizes the open and scalable standards of XML and SOAP.

In the rest of this chapter we initially present the technologies that constitute the Web services framework and are at the core of the WebDisC system. We then give background information on classification, classifier fusion and related work on distributed classifier systems. Subsequently, we describe the architecture, main functionality and user interface of the WebDisC system along with the X-DEVICE component of the system and the proposed classifier clustering approach. Finally, we conclude this work and pose future research directions.

WEB SERVICES

A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols (Champion et al., 2002).

The use of the Web services paradigm is expanding rapidly to provide a systematic and extensible framework for application-to-application (A2A) interaction, built on top of existing Web protocols and based on open XML standards. Web services aim to simplify the process of distributed computing by defining a standardized mechanism to describe, locate, and communicate with online software systems. Essentially, each application becomes an accessible Web service component that is described using open standards.

The basic architecture of Web services includes technologies capable of:

- Exchanging messages.
- Describing Web services.
- Publishing and discovering Web service descriptions.

Exchanging Messages

The standard protocol for communication among Web services is the Simple Object Access Protocol (SOAP) (Box et al., 2000). SOAP is a simple and lightweight XML-based mechanism for creating structured data packages that can be exchanged between network applications. SOAP consists of four fundamental components: an envelope that defines a framework for describing message structure, a set of encoding rules for expressing instances of application-defined data types, a convention for representing remote procedure calls and responses, and a set of rules for using SOAP with HTTP. SOAP can be used with a variety of network protocols, such as HTTP, SMTP, FTP, RMI/IIOP, or a proprietary messaging protocol.

SOAP is currently the de facto standard for XML messaging for a number of reasons. First, SOAP is relatively simple, defining a thin layer that builds on top of existing network technologies such as HTTP that are already broadly implemented. Second, SOAP is flexible and extensible in that rather than trying to solve all of the various issues developers may face when constructing Web services, it provides an extensible, composable framework that allows solutions to be incrementally applied as needed. Thirdly, SOAP is based on XML. Finally, SOAP enjoys broad industry and developer community support.

SOAP defines four XML elements:

- *env:Envelope* is the root of the SOAP request. At the minimum, it defines the SOAP namespace. It may define additional namespaces.
- *env:Header* contains auxiliary information as SOAP blocks, such as authentication, routing information, or transaction identifier. The header is optional.

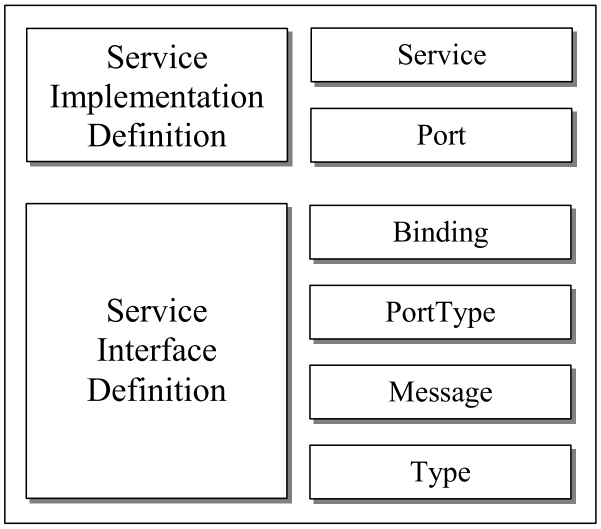
- *env:Body* contains one or more SOAP blocks. An example would be a SOAP block for RPC call. The body is mandatory and it must appear after the header.
- *env:Fault* is a special block that indicates protocol-level errors. If present, it must appear in the body.

SOAP is used in WebDisC for the exchange of messages between the Portal and the distributed classifiers. Examples of those messages can be found in Figures 9, 10 and 11.

Describing Web Services

The standard language for formally describing Web services is Web Services Description Language (WSDL). WSDL (Chinnici et. al, 2002), is an XML document format for describing Web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented (RPC) messages. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints may be combined into services. WSDL is sufficiently extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. A complete WSDL definition of a service comprises a service interface definition and a service implementation definition, as depicted in Figure 1.

Figure 1: WSDL service implementation and interface definitions


Service Implementation Definition
Service
Port
Service Interface Definition
Binding
PortType
Message
Type

A service interface definition is an abstract or reusable service definition that may be instantiated and referenced by multiple service implementation definitions. A service interface definition can be thought of as an IDL (Interface Definition Language), Java interface, or Web service type. This allows common industry standard service types to be defined and implemented by multiple service implementers.

In WSDL, the service interface contains elements that comprise the reusable portion of the service description: *binding*, *portType*, *message* and *type* elements. In the *portType* element, the operations of the Web service are defined. The operations define what XML messages can appear in the input, output and fault data flows. The *message* element specifies which XML data types constitute various parts of a message. The *message* element is used to define the abstract content of messages that comprise an operation. The use of complex data types within the message is described in the *type* element. The *binding* element describes the protocol, data format, security and other attributes for a particular service interface (*portType*).

The service implementation definition describes how a particular service interface is implemented by a given service provider. It also describes its location so that a requester can interact with it. In WSDL, a Web service is modeled as a *service* element. A *service* element contains a collection of *port* elements. A port associates an endpoint (e.g. a network address location) with a *binding* element from a service interface definition.

Examples of WSDL definitions for all WebDisC Web services can be found in (WebDisC, 2003).

Publishing and Discovering Web Service Descriptions

While there are some established standards for Web service description and communication, the publishing and discovery of Web services can be implemented with a range of solutions. Any action that makes a WSDL document available to a requestor, at any stage of the service requestor's lifecycle, qualifies as service publication. In the same way, any mechanism that allows the service requestor to gain access to the service description and make it available to the application at runtime qualifies as service discovery.

The simplest case of publishing a Web service is a direct publish. This means that the service provider sends the service description directly to the service requestor. This can be accomplished using an email attachment, an FTP site, or even a CDROM distribution. Slightly more dynamic publication uses Web Services Inspection Language (WSIL) (Brittenham, 2001). WSIL defines a simple HTTP GET mechanism to retrieve Web services descriptions from a given URL. Another means of publishing service descriptions available to Web services is through a Universal Description, Discovery and Integration (UDDI) registry (Bellwood et al., 2002). There are several types of UDDI registries that may be used depending on the scope of the domain of Web services published to it. When publishing a Web service description to a UDDI registry, complete business context and well though out taxonomies are essential if the service is to be found by its potential consumers.

The X-DEVICE system (Bassiliades et al., 2003a) is used in WebDisC, for registering and discovering Web services for distributed classification. More details are given later in the chapter.

CLASSIFIER FUSION: METHODS AND SYSTEMS

This section introduces the learning task of classification, the rationale for classifier fusion, established methods and existing systems that perform classifier fusion.

Classification

Supervised classification is one of the most common machine learning and data mining tasks (Saitta, 2000). It deals with the problem of identifying interesting regularities between a number of independent variables and a target or dependent categorical variable in a given data set. For example, given a set of training instances (x_{i1}, x_{i2}, …, x_{ik}, y_i), $i = 1..N$, the task is to compute a classifier, or model, or concept that approximates an unknown function y=f(x) that correctly labels any instance drawn from the same source as the training set.

There exist many ways to represent a classification model and many more algorithms to generate it. Typical classifier learning approaches include concept learning, neural networks, decision trees, rule learning, Bayesian learning and instance-based learning. All these approaches construct models that share the common ability to classify previously unknown instances of a domain based on instances of the same domain that were used for their training.

The output of a classifier can be i) the label of a class, ii) rankings for all the classes and iii) measures of uncertainty such as belief, confidence, probability, possibility, plausibility or other for each class. Consider for example, a domain for predicting tomorrow's weather with three possible classes: sunny, windy, rainy. The corresponding output for the three types of classifiers could be: i) sunny, ii) 1 - sunny, 2 - windy, 3 - rainy and iii) 0.8 - sunny, 0.5 - windy, 0.1 - rainy. Classifiers that output class labels are commonly called hard classifiers, while classifiers that output measures of uncertainty are called distribution/soft classifiers. Classifiers that output rankings are not so common in the machine learning literature.

Another distinction among classifiers is whether they are homogeneous or heterogeneous. There are two forms of classifier heterogeneity. According to the first, two classifiers are considered homogeneous if they are created using the same learning algorithm. For example a naive Bayes classifier and a decision list are heterogeneous classifiers, while two neural networks are homogeneous classifiers. Another form of heterogeneity is based on the schema of the training data of the two classifiers. For example, two decision trees that both predict tomorrow's weather, but one is based on temperature and wind speed, while the other on atmospheric pressure and humidity are considered heterogeneous classifiers. In this chapter, the term heterogeneity will be used with the latter meaning.

Fusion Methods

Classifier Fusion has been a very active field of research in the recent years. It was used for improving the classification accuracy of pattern recognition systems, as single classification learning algorithms were approaching their limits. It was also used as a method for scaling up data mining to very large databases, through combining classifiers trained in parallel from different parts of the database. Finally, it was used for learning from geographically distributed databases, where bandwidth or privacy constraints forbid the gathering of data in a single place, through the fusion of locally learned classification models.

There are two general groups of Classifier Fusion methods. The first group encompasses methods that combine the outputs of the classifiers, while the second group deals with the structure of the multiple classifier system. We will focus on the former group of methods, as WebDisC implements some of them and provides the necessary infrastructure for implementing the rest.

Methods that fuse classifier outputs can be further categorized based on two properties. The first is the classifier output type on which they can operate, and the second is the need of training data for the fusion process. According to these, Table 1 presents the main methods. WebDisC currently implements the simple methods of Majority Voting and the Sum, Product, Min, Max and Median rules.

Table 1: Classifier fusion methods

	Re-Training	
Output	Yes	No
Label	Knowledge-Behavior Space (Huang & Suen, 1995)	Majority Voting (Lam & Suen, 1995)
Ranking	The Highest Rank Logistic Regression Intersection of Neighborhoods Union of Neighborhoods	Borda Count
Distribution	Stacked Generalization (Wolpert, 1992) Dempster-Shaffer Combination (Rogova, 1994) Fuzzy Templates (Kuncheva et al., 1995) Fuzzy Integrals (Tahani & Keller, 1990)	Sum, Product, Min, Max, Median rules (Kittler et al., 1998)

Majority Voting works for both hard and distribution classifiers. In the latter case the class with the maximum certainty measure receives one vote, breaking ties arbitrarily. The Sum, Min, Max, Prod and Median rules apply to distribution classifiers only. An interesting study of these rules for classifier combination can be found in (Kittler et al., 1998).

Let $C = \{C_1, C_2, ..., C_N\}$ be a set of classifiers and $L = \{L_1, L_2,..., L_K\}$ be a set of class labels. Hard classifiers receive an input instance x and output an element of the set L. Distribution classifiers receive an input instance x and output a k-dimensional vector $C_i(x) = [c_{i,1}(x), c_{i,2}(x), ..., c_{i,k}(x)]$, where $c_{i,j}(x)$ is the certainty measure that classifier C_i gives to label L_j.

For the Majority Voting combination of hard classifiers, the output is the element of set L that got the most votes (outputs) from the N classifiers. For the Sum, Min, Max, Prod and Median rules the output is a k-dimensional vector $[r_1(x), r_2(x),..., r_k(x)]$, where:

$$r_i(x) = op(c_{1,i}(x), c_{2,i}(x), ..., c_{N,i}(\mathrm{I}))$$

and op is the respective operation (average, minimum, maximum, product and median).

Fusion Systems

Despite the availability of many classifier fusion methods, there are few systems that implement such methods in a distributed database scenario. A reason is that most of these methods were used for pattern recognition tasks, where data are usually gathered at a single place and there are no distributed computing requirements. In the following paragraphs we summarize some of the most important work on system development aimed at classifier learning and combination from distributed data.

A system that learns and combines classifiers from distributed databases is Java Agents for Meta-Learning (JAM) (Stolfo et al., 1997). It is implemented in Java and uses the Java technology of Remote Method Invocation (RMI) for distributed computing. An important limitation to the extensibility of the system is the fact that clients have to be written in Java. Therefore, JAM is bound to be a closed system that is intended for use in a group of firmly-coupled distributed databases. Furthermore, in contrast to WebDisC, it cannot be used for the fusion of heterogeneous classifiers.

A CORBA infrastructure for distributed learning and meta-learning is presented in (Werges & Naylor, 2002). Although CORBA is a standard-based distributed object technology, is has been surpassed by the Web services technology. In addition, the presented infrastructure is very complex as the client developers have to implement a lot of different interfaces. The lack of open standards such as XML and SOAP and the complexity of the system hinder its extensibility. Furthermore, like JAM it combines homogeneous classifiers.

Another system that is built using CORBA technology is MAS (Botia et al., 2001). This is a sophisticated system with some interesting features that add to its extensibility. These include a common services interface for all learning agents and an X.500-based directory service as a repository for the system components. However, as stated above, these standards have been surpassed by Web services technologies which are open, scalable and extensible.

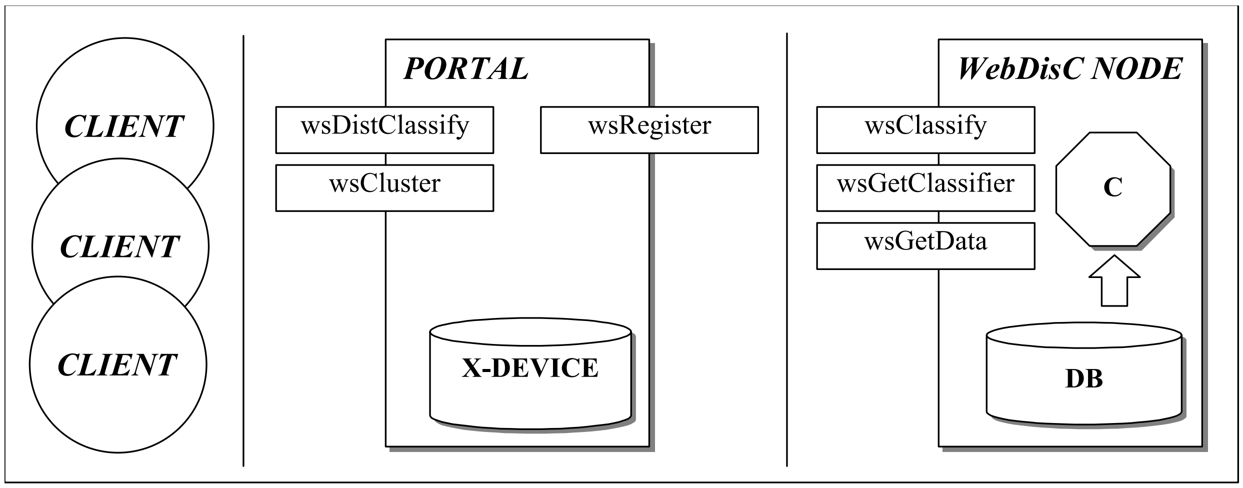
THE WEBDISC SYSTEM

WebDisC is a knowledge-based Web information system for the fusion of classifiers induced at geographically distributed databases. The architecture of WebDisC is based on the new and promising Web services paradigm. It further encompasses a Web-based interface that allows users to interact with the system through guided procedures. It's main functionality includes: i) a declarative rule language for classifier selection that allows the combination of syntactically heterogeneous distributed classifiers, ii) a variety of standard methods for fusing the output of distributed classifiers and iii) a new approach for clustering classifiers in order to deal with the semantic heterogeneity of distributed classifiers, detect their interesting similarities and differences and enhance their fusion. The rest of this section describes the architecture, functionality, methodologies and user interface of the system.

Architecture and Main Functionality

The architecture of WebDisC comprises 3 basic components as depicted in Figure 2: i) Clients, ii) The Portal and iii) WebDisC Nodes.

Figure 2: The architecture of WebDisC



The WebDisC Nodes

WebDisC Nodes are databases located at different geographical areas along with local classification models that were induced from those databases using a machine learning/data mining algorithm.

WebDisC Nodes expose the following web services:

- *wsClassify* takes as input the attribute-value pairs of an unclassified example and outputs the classification result.
- *wsGetClassifier* takes an empty input and returns the classifier of the WebDisC Node in PMML format (Data Mining Group, 2002).
- *wsGetData* returns a vector of tuples from the WebDisC Node's database. It takes as input an integer indicating the number of tuples that will be transferred.

Notice that the WSDL descriptions for all the Web services of our system can be found in (WebDisC, 2003).

The Portal

The Portal is the coordinating component of the system. It consists of the X-DEVICE deductive XML database system and the following web services: *wsRegister*, *wsDistClassify* and *wsCluster*. In addition, it offers a Web-based interface for thin client access, that also implements the fusion methods.

X-DEVICE's main purpose is the storage of meta-data regarding the distributed classifiers that are registered with WebDisC. These meta-data include: description, names and types of the input and output attributes, name of learning algorithm, ability to handle missing values and the URI of the Web services.

Figure 3 shows the DTD for the classifier's meta-data, which also define the type of objects that are stored in X-DEVICE for each classifier, according to the XML-to-object mapping scheme of X-DEVICE (see Figure 8). Notice that the actual XML Schema1 data types for *attType* and *address* elements are *xs:anyType* and *xs:anyURI*, respectively. Figure 4 shows sample metadata for a classifier registered in X-DEVICE, according to the DTD of Figure 3. More on X-DEVICE will be presented in the corresponding section.

Figure 3: DTD for classifier metadata

```
<!ELEMENT classifier (name, description, address,
                      classificationMethod, acceptsMissingValues,
                      classificationAttribute, inputAttribute*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT classificationMethod (#PCDATA)>
<!ELEMENT acceptsMissingValues (#PCDATA)>
<!ELEMENT classificationAttribute (attName, attName)>
<!ELEMENT inputAttribute (attName, attType)>
<!ELEMENT attName (#PCDATA)>
<!ELEMENT attType (#PCDATA)>
```

Figure 4: Sample classifier metadata

```
<classifier>
  <name>Classifier1</name>
  <description>A local classifier that uses a Decision Tree</description>
  <address>http://startrek.csd.auth.gr/Classifier1</address>
  <classificationMethod>Decision Tree</classificationMethod>
  <acceptsMissingValues>true</acceptsMissingValues>
  <classificationAttribute>
    <attName>loan</attName>
    <attType>xs:string</attType>
  </classificationAttribute>
  <inputAttribute>
    <attName>income</attName>
    <attType>xs:integer</attType>
  </inputAttribute>
  <inputAttribute>
    <attName>card</attName>
    <attType>xs:string</attType>
  </inputAttribute>
  <inputAttribute>
    <attName>home</attName>
    <attType>xs:string</attType>
  </inputAttribute>
</classifier>
```

wsRegister is the web service that WebDisC Nodes use in order to register with the system. This service takes as input the classifier meta-data of a WebDisC Node and adds them within X-DEVICE (see Figure 9).

wsDistClassify, implements a new approach for distributed classification. It takes as input the name of the dependent attribute and the names and values of some independent attributes of an unclassified example. It also takes as input the name of a predefined method or a set of user-defined X-DEVICE rules that specify a strategy for selecting the classifiers that should be used for classifying this example, amongst all suitable classifiers. The selection strategies offered by the system along with the specifications for describing user-defined strategies are further explained on page 21. The service retrieves from X-DEVICE the URIs and SOAP messages of the selected classifiers and calls the *wsClassify* web service of the distributed classifiers passing the names and values of the corresponding independent attributes as arguments. The output of the service is a list of the collected results.

wsCluster, implements a new approach for clustering distributed classifiers. It takes as input a list of URIs that correspond to a group of *N* classifiers that all share the same input and output attributes and calls their *wsGetData* and *wsGetClassifier* services. It so retrieves the actual classifiers and necessary data to centrally run the clustering algorithm that is explained in the C*lustering Distributed Classifiers* section. The output of the service is the clustering result in the form of a vector of size *N* with numbers indicating the cluster of each classifier.

Finally, the Portal offers a Web-based user interface with guided procedures for using the system. Users of WebDisC can select from the main page one of the tasks of classification or clustering and are directed to the corresponding pages. Data are entered via dynamically created HTML forms from classifier meta-data stored in the X-DEVICE system. Classification and clustering results are also presented in the Web-browser. Java servlets handle the form input data and the preparation of results. The user interface is detailed in the following subsection.

The Clients

Thick clients (applications) that want to exploit the functionality of WebDisC can directly use the Portal's *wsDistClassify* and *wsCluster* Web services. In addition, thin clients (Web browsers) can access the functionality of WebDisC by visiting the system's web pages and perform one of the tasks through guided procedures.

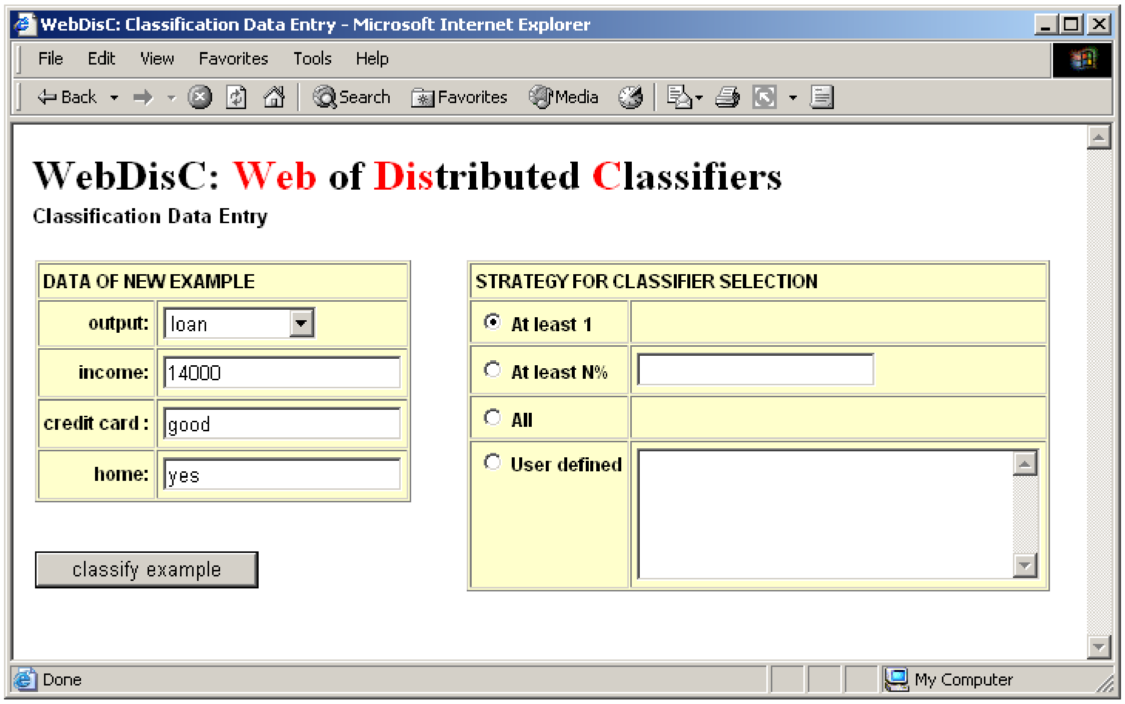
User Interface

The main Web page of WebDisC allows users to select either the main task of classification or that of clustering.

Classification

The classification data entry page contains a form where users can fill in the details of the example to be classified and a form for selecting or entering the classifier selection strategy. Figure 5 shows this page completed with values for a specific domain, regarding the approval of a loan application.

Figure 5: Example of the classification data entry Web page of WebDisC


WebDisC: Classification Data Entry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Done
My Computer
classify example

WebDisC: Web of Distributed Classifiers

Classification Data Entry

DATA OF NEW EXAMPLE	
output:	loan
income:	14000
credit card :	good
home:	yes

STRATEGY FOR CLASSIFIER SELECTION	
● At least 1	
○ At least N%	
○ All	
○ User defined	

In the general case, users first select one of the available output attributes from a combo-box. The entries of this combo-box are dynamically created from the meta-data of the registered classifiers using a simple X-DEVICE query. Once an output attribute has been selected, the page reloads with a table containing the names of all the input attributes that can be used to predict this output attribute. Again this is performed dynamically through a simple X-DEVICE query. Users can fill in the values for these attributes in the corresponding text-boxes. If an attribute value is left unspecified then it is ignored.

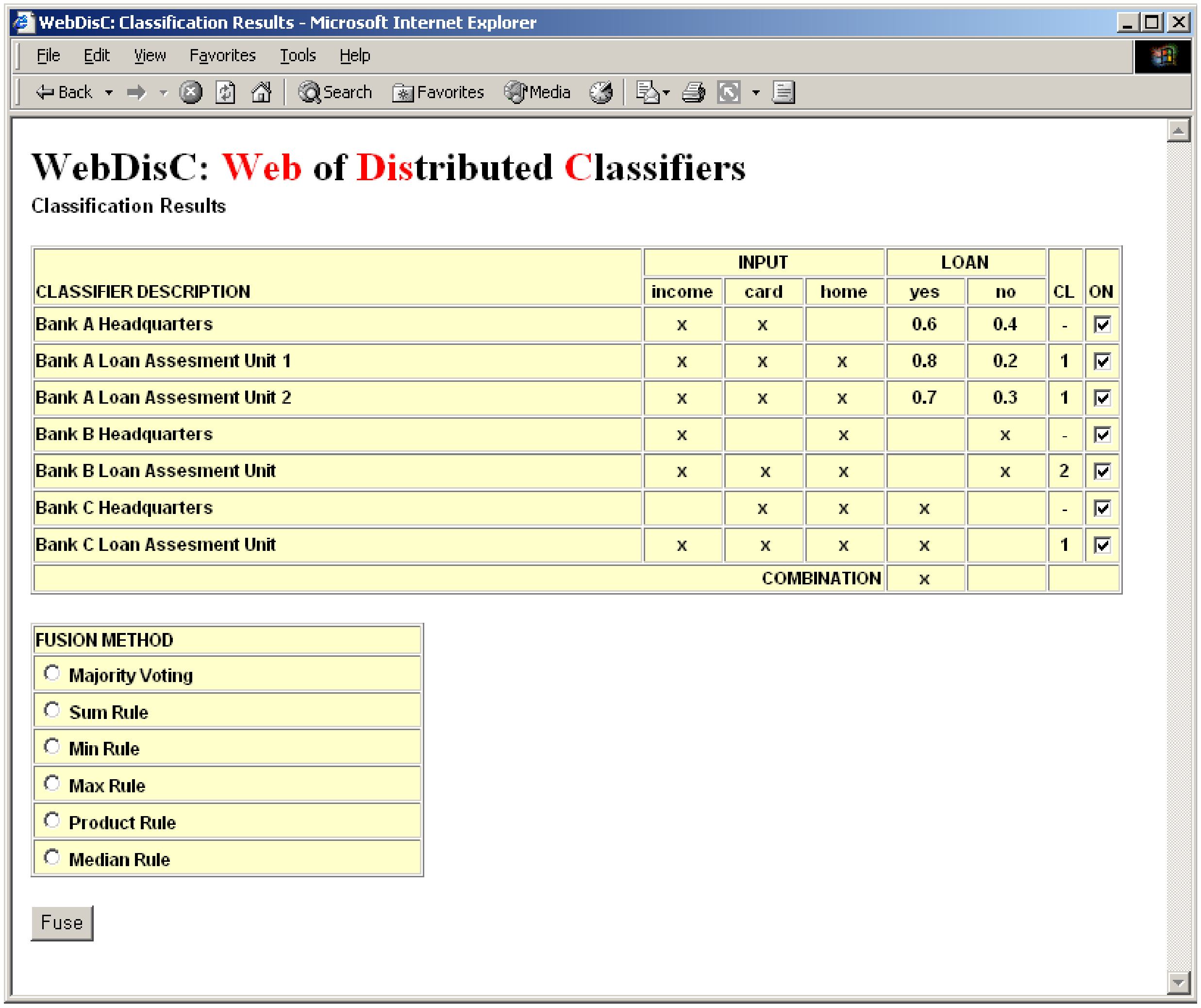
After entering the data of the example, users can select one of the default classifier selection strategies or enter their own using the X-DEVICE query language. The default strategies are: i) Select the classifiers that have at least one input attribute in common with the attributes of the new example, ii) select the classifiers that have at least N% of their input attributes in common with the attributes of the new example and iii) select the classifiers that have all their input attributes in common with the attributes of the new example. This last strategies select homogeneous classifiers.

After selecting a strategy, users can press the *classify example* button, which calls the *wsClassify* service of the Portal through a Java servlet passing the entered data and the selection strategy as arguments. The classification results that the *wsClassify* service outputs are subsequently visualized on another Web page by a table with a row for each classifier result. An example of such a table filled in with values for the loan application example is depicted in Figure 6.

In this example, 7 classifiers from 3 different banks registered with WebDisC match the input data that the user entered. The loan assessment units of the banks have homogeneous classifiers that take as input all three attributes, while the headquarters of each bank have classifiers that use a different combination of input attributes. Furthermore, we notice only bank A uses classifiers that output distributions, while the rest classifiers output class labels. Another thing that can be noticed is that banks A and C output a *yes* decision to the loan application, while bank B classifiers output a *no* decision. Column *CL* of the table concerns the clustering process and will be subsequently explained.

At the bottom of the page there is a fuse button, which users can press for combining the results of classifiers. The users can select one of the five supported combination methods as explained in the *Classifier Fusion:Methods and Systems* section. By default, all classifiers are selected for participating in the combination. To exclude one or more classifiers, users can uncheck the check box in the last column (*ON*) of the table. When the *fuse* button is pressed the page reloads with the fusion result presented in the final row of the classifier results table, as depicted in Figure 6. In this example, the Majority Voting decision is *yes* as five out of the 7 classifiers output *yes*. If one of the other fusion methods were selected, then the classifiers of banks B and C would not have taken part in the fusion, because they are not distribution classifiers, as these methods demand.

Figure 6: Example of the classification results Web page of WebDisC


WebDisC: Classification Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media

WebDisC: Web of Distributed Classifiers

Classification Results

CLASSIFIER DESCRIPTION	INPUT			LOAN		CL	ON
	income	card	home	yes	no		
Bank A Headquarters	x	x		0.6	0.4	-	✓
Bank A Loan Assesment Unit 1	x	x	x	0.8	0.2	1	✓
Bank A Loan Assesment Unit 2	x	x	x	0.7	0.3	1	✓
Bank B Headquarters	x		x		x	-	✓
Bank B Loan Assesment Unit	x	x	x		x	2	✓
Bank C Headquarters		x	x	x		-	✓
Bank C Loan Assesment Unit	x	x	x	x		1	✓
COMBINATION				x			

FUSION METHOD
○ Majority Voting
○ Sum Rule
○ Min Rule
○ Max Rule
○ Product Rule
○ Median Rule

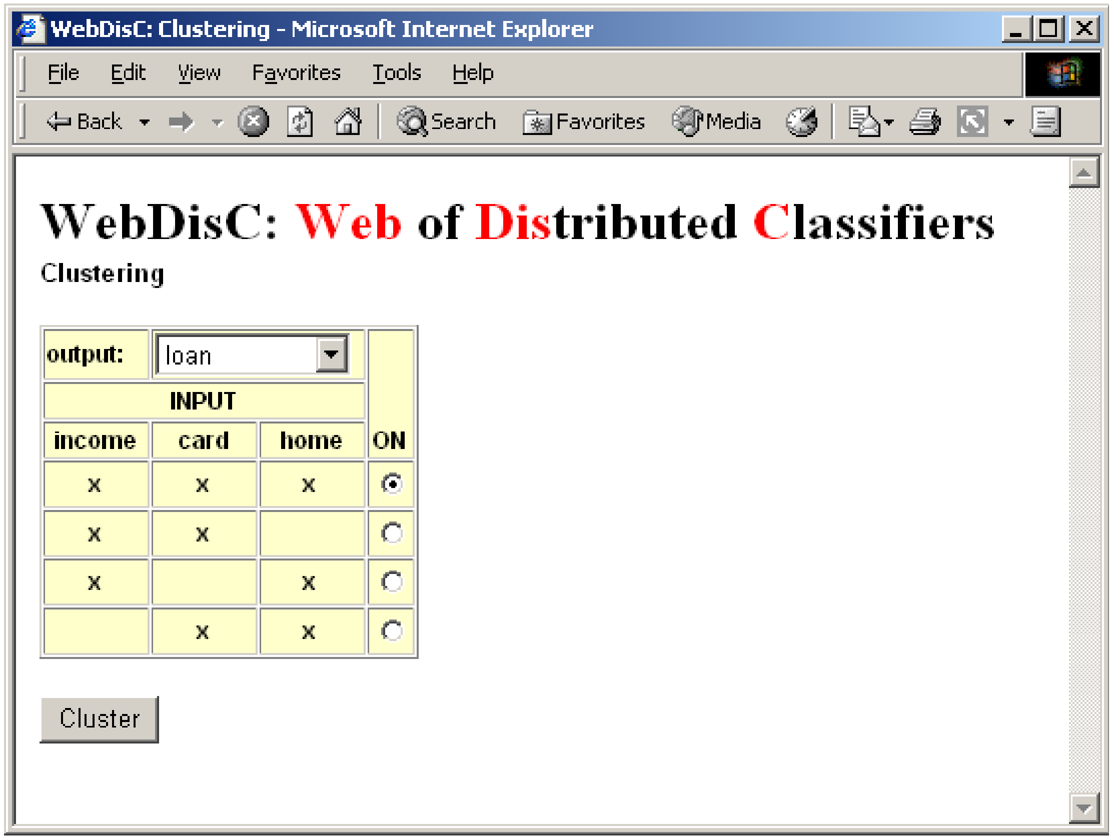
Fuse

Clustering

The clustering page contains a combo-box for selecting the output attribute similarly to the classification page. Upon selecting this attribute the page reloads with a table that has a row for each group of homogeneous classifiers that can predict it. The content of this table is dynamically calculated through a simple query to X-DEVICE. Each row describes the input attributes of the classifier group and has a radio button next to it so that only one group can be selected.

Figure 7 shows the clustering page for the loan application example of the previous section. There are four groups of homogeneous classifiers. The first corresponds to classifiers from the four loan assessment units, while the rest from each of the bank headquarters.

Figure 7: Example of the clustering Web page of WebDisC


WebDisC: Clustering - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Cluster

WebDisC: Web of Distributed Classifiers

Clustering

output:	loan		
INPUT			
income	card	home	ON
x	x	x	[✓ ●]
x	x		○
x		x	○
	x	x	○

Pressing the cluster button at the bottom of the page will call the *wsCluster* service of the Portal passing as parameters the URLs of the classifiers that belong to the selected group. The result of the clustering process will be stored along with the classifier meta-data in X-DEVICE for future use in a classification process. Going back to Figure 6, we can see the result of clustering the four loan assessment units of the banks. Column CL of the table has as value for each classifier the number of the cluster to which it belongs. We can see that there are two groups. One contains the classifiers from the loan assessment units of bank A and C and the other the classifier from the loan assessment unit of bank B.

Classifier clustering aims at discovering semantic differences that arise from the geographical distribution of the databases. Using the above example, banks A and C could belong to poorer countries than bank B and thus consider the income of the example as high enough for the loan to be granted. Bank B however, considers the income low for granting the loan. In this case bank B classifiers should never be fused with bank A and C classifiers due to semantic differences of the classification models. Having performed the clustering process, the user of WebDisC has gained this knowledge and can avoid fusing classifiers belonging to different clusters. Details of the clustering algorithm can be found in the *Clustering Distributed Classifiers* section.

THE X-DEVICE COMPONENT

In this section, we initially present an overview of the X-DEVICE system, a deductive object-oriented XML database system (Bassiliades et al., 2003a) that is used as a Web service registry component for the WebDisC system. Then, we describe in detail the functionality of X-DEVICE within WebDisC.

Overview of the X-DEVICE System

In X-DEVICE, XML documents are stored into the OODB by automatically mapping the DTD to an object schema. Furthermore, X-DEVICE employs a powerful rule-based query language for intelligently querying stored Web documents and data and publishing the results. X-DEVICE is an extension of the active object-oriented knowledge base system DEVICE (Bassiliades et al., 2000). DEVICE integrates deductive and production rules into an active OODB with event-driven rules (Diaz & Jaime, 1997), on top of Prolog. This is achieved by translating the condition of each declarative rule into a set of complex events that is used as a discrimination network to incrementally match the condition against the database.

The advantages of using a logic-based query language for XML data come from the well-understood mathematical properties and the declarative character of such languages, which both allow the use of advanced optimization techniques, such as magic-sets. Furthermore, X-DEVICE compared to the Xquery (Boag et al., 2002) functional query language has a more high-level, declarative syntax that allows users to express everything that XQuery can express, in a more compact and comprehensible way, with the powerful addition of general path expressions, which is due to fixpoint recursion and second-order variables.

XML Object Model

The X-DEVICE system translates DTD definitions into an object database schema that includes classes and attributes, while XML data are translated into objects. Generated classes and objects are stored within the underlying object-oriented database ADAM (Gray et al., 1992). The mapping of a DTD element to the object data model depends on the following:

- If an element has `PCDATA` content (without any attributes), it is represented as a string attribute of the class of its parent element node.The name of the attribute is the same as the name of the element.
- If an element has either a) children elements, or b) attributes, then it is represented as a class that is an instance of the `xml_seq` meta-class. The attributes of the class include both the attributes of the element and the children elements. The types of the attributes of the class are determined as follows:

- Simple character children elements and element attributes correspond to object attributes of string type. Attributes are distinguished from children elements through the `att_lst` meta-attribute.
- Children elements that are represented as objects correspond to object reference attributes.

The order of children elements is handled outside the standard OODB model by providing a meta-attribute (`elem_ord`) for the class of the element that specifies the correct ordering of the children elements. This meta-attribute is used when (either whole or a part of) the original XML document is reconstructed and returned to the user. The query language also uses it.

Alternation is also handled outside the standard OODB model by creating a new class for each alternation of elements, which is an instance of the `xml_alt` meta-class and it is given a unique system-generated name. The attributes of this class are determined by the elements that participate in the alternation. The structure of an alternation class may seem similar to a normal element class; however the behavior of alternation objects is different, because they must have a value for exactly one of the attributes specified in the class.

The mapping of the multiple occurrence operators, such as "star" (*), etc, are handled through multi-valued and optional/mandatory attributes of the object data model. The order of children element occurrences is important for XML documents, therefore the multi-valued attributes are implemented as lists and not as sets.

Figure 8 shows the X-DEVICE representation of the XML document in Figure 4

Figure 8: X-DEVICE representation of the XML document in Figure 4

```
object          1#classifier
  instance      classifier
  attributes
  name                          'Classifier1'
    description                 'A local classifier that uses a Decision Tree'
    address                     'http://startrek.csd.auth.gr/Classifier1'
    classificationMethod        'Decision Tree'
    acceptsMissingValues        true
    classificationAttribute     2#classificationAttribute
    inputAttribute              [3#inputAttribute,4#inputAttribute,5#inputAttribute]

object          2#classificationAttribute    object          3#inputAttribute
  instance      classificationAttribute        instance      inputAttribute
  attributes                                   attributes
    attName             loan                     attName             income
    attType             xs:string                attType             xs:integer

object          4#inputAttribute             object           5#inputAttribute
  instance      inputAttribute                 instance       inputAttribute
  attributes                                   attributes
    attName             card                     attName             home
    attType             xs:string                attType             xs:string
```

XML Deductive Query Language

X-DEVICE queries are transformed into the basic DEVICE rule language and are executed using the system's basic inference engine. The query results are returned to the user in the form of an XML document. The deductive rule language of X-DEVICE supports generalized path and ordering expressions, which greatly facilitate the querying of recursive, tree-structured XML data and the construction of XML trees as query results. These advanced expressions are implemented using second-order logic syntax (i.e. variables can range over class and attribute names) that have also been used to integrate heterogeneous schemata (Bassiliades et al., 2003b). These XML-aware constructs are translated through the use of object meta-data into a combination of a) a set of first-order logic deductive rules, and/or b) a set of production rules that their conditions query the meta-classes of the OODB, they instantiate the second-order variables, and they dynamically generate first-order deductive rules.

In this section we mainly focus on the use of the X-DEVICE first-order query language to declaratively query the meta-data of the classifier Web services that are represented as XML documents. More details about DEVICE and X-DEVICE can be found in (Bassiliades et al., 2000) and (Bassiliades et al., 2003a). The general algorithms for the translation of the various XML-aware constructs to first-order logic can be found in (X-DEVICE, 2002).

In X-DEVICE, deductive rules are composed of condition and conclusion, whereas the condition defines a pattern of objects to be matched over the database and the conclusion is a derived class template that defines the objects that should be in the database when the condition is true. For example, rule R4 (see next subsection) defines that an object with attribute `classifierID` with value `CL` and attribute `address` with value `URL` exists in class `candidate_classifier` if there is an object with OID `I` in class `init_candidate` with an attribute `method` that its value equals string "At least one", an attribute `address` with value `URL` and an attribute `classifierID` with value `CL`, which points to an object of class `classifier` which in turn has an attribute `acceptsMissingValues` with value "true".

Actually, rule R4 selects all the initial candidate classifiers if the selection method requires at least one input attribute common to the user's classification request and the classifier accepts missing values for some of its input attributes. Class `candidate_classifier` is a derived class, i.e. a class whose instances are derived from deductive rules. Only one derived class template is allowed at the THEN-part (head) of a deductive rule. However, many rules can exist with the same derived class at the head (e.g. rules R15 and R16). The final set of derived objects is a union of the objects derived by all the rules.

The syntax of such a rule language is first-order. Variables can appear in front of class names (e.g. `I`, `CL`), denoting `OID`s of instances of the class, and inside the brackets, denoting attribute values, i.e. object references (`CL`) and simple values (`URL`), such as integers, strings, etc. Variables are instantiated through the ":" operator when the corresponding attribute is single-valued, and the "϶" operator when the corresponding attribute is multi-valued. Conditions can also contain comparisons between attribute values, constants and variables. Negation is also allowed if rules are safe, i.e. variables that appear in the conclusion must also appear at least once inside a non-negated condition.

Path expressions can be composed using dots between the "steps", which are attributes of the interconnected objects, which represent XML document elements. For example rule R2 generates the set of initial candidate classifiers by selecting all the registered classifiers that have at least one input attribute common to the user's classification request. The object that represents the user's request is `C@classify` and in order to retrieve names of input attributes the query navigates from `classify` through `inputVector` and `inputPair` to `attName`.

The innermost attribute should be an attribute of "departing" class, i.e. `inputVector` is an attribute of class `classify`. Moving to the left, attributes belong to classes that represent their predecessor attributes. Notice the right-to-left order of attributes, contrary to the common C-like dot notation, that stress out the functional data model origins of the underlying ADAM OODB (Gray et al., 1992). Under this interpretation the chained "dotted" attributes can be seen as function compositions.

A query is executed by submitting the set of stratified rules (or logic program) to the system, which translates them into active rules and activates the basic events to detect changes at base data. Data are forwarded to the rule processor through a discrimination network (much alike in a production system fashion). Rules are executed with fixpoint semantics (semi-naive evaluation), i.e. rule processing terminates when no more new derivations can be made. Derived objects are materialized and are either maintained after the query is over or discarded on user's demand. X-DEVICE also supports production rules, which have at the THEN-part one or more actions expressed in the procedural language of the underlying OODB.

The main advantage of the X-DEVICE system is its extensibility; it allows the easy integration of new rule types as well as transparent extensions and improvements of the rule matching and execution phases. The current system implementation includes deductive rules for maintaining derived and aggregate attributes. Among the optimizations of the rule condition matching is the use of a RETE-like discrimination network, extended with reordering of condition elements, for reducing time complexity and virtual-hybrid memories, for reducing space complexity (Bassiliades & Vlahavas, 1997). Furthermore, set-oriented rule execution can be used for minimizing the number of inference cycles (and time) for large data sets (Bassiliades et al., 2000).

More examples of the X-DEVICE language will be presented and explained in the sequel when needed for the description of the WebDisC functionality.

X-DEVICE Functionality in WebDisC

In this subsection we describe in detail the functionality of the X-DEVICE system within the WebDisC system, as it has been previously presented in the WebDisC system architecture.

Classifier Registration

The initial task that X-DEVICE performs within WebDisC is to register the meta-data for the classifiers of the WebDisC nodes. The DTD of the classifiers' meta-data has been given in Figure 3. The WSDL description for the *wsRegister* service is shown in (WebDisC, 2003). New WebDisC nodes sent in a SOAP message that contains their meta-data. A sample SOAP message is shown in Figure 9. The schema of the incoming SOAP message is determined at the input message of the corresponding port type of the WSDL description.

Figure 9: Sample SOAP message for registering a classifier

Input SOAP messages are stored within the X-DEVICE system using the schema for the SOAP message found in the corresponding WSDL description. However, the top-level element node of the input

SOAP message is linked to an instance of the `input_soap_message` class, through the OID of the object-element node and its attribute content.

The following X-DEVICE rule Rl iterates over all incoming SOAP messages that register a new classifier and generates a new classifier object for each one of them.

```
Rl:
if   I@input_soap_message(content:R) and
     R@register(classifierName:Name,classifierDescription:Desc,
                address:Address,classificationMethod:Method,
                acceptsMissingValues:AMV,
                classificationAttribute:CA,inputAttribute:IA)
then classifier(name:Name,description:Desc,address:Address,
                classificationMethod:Method,acceptsMissingValues:AMV,
                classificationAttribute:CA,inputAttribute:IA)
```

Actually, rule Rl transforms the XML data of SOAP messages (Figure 9) into classifier metadata (Figure 4), stored as a set of objects (Figure 8).

Classifier Selection

One very important task of X-DEVICE is the selection of classifiers that are relative to the user's request. Initially, rule R2 below pre-selects all classifiers that have at least one input attribute `Att` common to the input SOAP message for the *wsDistClassify* service.

```
R2:
if   I@input_soap_message(classify:C) and
     C@classify(select:Method,classificationAtt:CAtt,
                attName.inputPair.inputVector:Att) and
     CL@classifier(address:URL,attName.inputAttribute=Att,
                   attName.classificationAttribute=CAtt)
then init_candidate(method:Method,classifierID:CL,address:URL)
```

Figure 10 shows an example of such a SOAP message. Notice that the classification attribute `CAtt` of the registered classifier must also coincide with the requested classification attribute. The selection strategy `Method` provided by the user is kept along the initial set of candidate classifiers in order to be used for the next step of classifier selection.

Furthermore, all the input attributes of the input SOAP message that match some of the initial set of candidate classifiers are also kept as instances of the `candidate_atts` class, using rule R3.

```
R3:
if    I@input_soap_message(inputPair.inputVector.classify ∍ P) and
      P@inputPair(attName:Att,attValue:Val) and
      IC@init_candidate(classifierID:CL) and
      CL@classifier(attName.inputAttribute=Att)
then  candidate_atts(classifierID:CL,attribute:Att,value:Val)
```

The above classes, `init_candidate` and `candidate_atts`, constitute the programming interface for the rules that implement the classifier selection strategy. Some of these strategies, such as *At* *least one*, *All*, *At least N%*, are provided by the system. A knowledgeable user can also use the X-DEVICE language to provide his/her own selection strategy.

Figure 10: Sample SOAP message for classifying an example through the wsDistClassify service

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:m0="http://startrek.csd.auth.gr/wsDistClassify.xsd">
  <SOAP-ENV:Body>
    <m:Classify xmlns:m="http://startrek.csd.auth.gr/wsDistClassify.wsdl">
      <inputVector>
        <inputPair>
         <m0:attName>income</m0:attName>
         <m0:attValue>14000</m0:attValue>
        </inputPair>
        <inputPair>
         <m0:attName>card</m0:attName>
         <m0:attValue>good</m0:attValue>
        </inputPair>
        <inputPair>
         <m0:attName>home</m0:attName>
         <m0:attValue>yes</m0:attValue>
        </inputPair>
      </inputVector>
      <classificationAtt>loan</classificationAtt>
      <select>At least one</select>
    </m:Classify>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Rule R4 below implements the selection of a classifier that has at least one common input attribute with the input SOAP message. Notice that the set of the initially selected candidate classifiers already satisfies the above requirement, therefore the following rule just checks if a classifier accepts missing input values and then copies its OID and address to the output interface class `candidate_classifier`.

```
R4:
if   I@init_candidate(method='At least one',classifierID:CL,address:URL) and
     CL@classifier(acceptsMissingValues='true')
then candidate_classifier(classifierID:CL,address:URL)
```

The selection of the classifiers that all their input attributes are present at the input SOAP message needs a more complicated treatment. Rule R5 iterates over all initial candidate classifiers and excludes the ones that have an input attribute not present at the input SOAP message (instances of `candidate_atts` class). Then, rule R6 copies to the `candidate_classifier` class the OID and address of the initial candidate classifiers that have not been excluded by rule R5.

```
R5:
if   I@init_candidate(method='All',attName.inputAttribute.classifierID:Att) and
     not C@candidate_atts(classifierID=CL,attribute=Att)
then exclude_candidate(classifier:I)


R6:
if   C@init_candidate(method='All',classifierID:CL,address:URL) and
     not C1@exclude_candidate(classifier=C)
then candidate_classifier(classifierID:CL,address:URL)
```

Finally, the selection of a classifier if the input SOAP message has at least N% of the input attributes of the classifier needs aggregate functions that count the total number of input attributes of the classifier (rule R7) and the total number of the input attributes of the classifier that are present at the input SOAP message (rule R8). Rule R9 retrieves the two numbers, calculates their ratio, and compares it to the user-supplied percentage. Notice that the selected classifier needs to accept missing values. All three rules make use of the `prolog{}` construct of X-DEVICE to call out arbitrary Prolog goals.

```
R7:
if   I@init_candidate(method=N,attName.inputAttributes.classifierID:Att) and
     prolog{number(N)}
then candidate_total_atts(classifier:I,atts_no:count(Att))


R8:
if   I@init_candidate(method=H,classifierID:CL) and
     C@candidate_atts(classifierID=CL,attribute:Att) and
     prolog{number(N)}
then candidate_existing_atts(classifier:I,atts_no:count(Att))


R9:
if   C@init_candidate(method=N,classifierID:CL,address:URL) and
     CL@classifier(acceptsMissingValues='true') and
     CT@candidate_total_atts(classifier=C,atts_no:Total) and
     CE@candidate_existing_atts(classifier=C,atts_no:Existing) and
     prolog{number(N),P is 100*Existing/Total,P>=N}
then candidate_classifier(classifierID:CL,address:URL)
```

The addresses of the final set of candidate classifiers are returned to the requesting application along with the corresponding SOAP messages that should be sent to the *wsClassify* services of the WebDisC nodes. Figure 11 shows such a message.

The result is returned as an XML document and is calculated by the rules R10 to R14. Rule R10 creates a classify object that points to a selected classifier object. Notice the use of the exclamation mark (!) in front of an attribute name to denote a system attribute, i.e. an auxiliary attribute that will not be a part of the query result. Rule Rll constructs a `classifyPair` object for each attribute-value pair of each selected classifier.

Figure 11: Sample SOAP message for classifying an example through the wsClassify service

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:m0="http://startrek.csd.auth.gr/wsClassify1.xsd">
  <SOAP-ENV:Body>
    <m:Classify xmlns:m="http://startrek.csd.auth.gr/wsClassify1.wsdl">
      <classifyVector>
        <classifyPair>
          <m0:attName>income</m0:attName>
          <m0:attValue>14000</m0:attValue>
        </classifyPair >
        <classifyPair >
          <m0:attName>card</m0:attName>
          <m0:attValue>good</m0:attValue>
        </classifyPair>
      </classifyVector>
    </m:Classify>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Rule R12 creates a `classifyVector` object for each selected classifier and links it with the corresponding `classifyPair` objects. The `list(CP)` construct in the rule conclusion denotes that the attribute `classifyPair` of the derived class `classifyVector` is an attribute whose value is calculated by the aggregate function `list`. This function collects all the instantiations of the variable `CP` (since many input attributes can exist for each classifier) and stores them under a strict order into the multi-valued attribute `classifyPair`. Notice that the values of the rest of the variables at the rule conclusion define a GROUP BY operation. More details about the implementation of aggregate functions in X-DEVICE can be found in (Bassiliades et al., 2000) and (Bassiliades et al., 2003a).

Rule R13 links the `classifyVector` object with the corresponding `classify` object through a derived attribute rule, which defines a new attribute `classifyVector` for class `classify`. The values for this attribute are derived by this rule. Objects of class `classify` that do not satisfy the condition of this class will have null value for this attribute. More details on derived attribute rules can be found in (Bassiliades et al., 2000).

Finally, rule R14 constructs the top-level XML element of the result which is the SOAP message built for each classifier, augmented with the address of the classifier. The keyword `xml_result` is a directive that indicates to the query processor that the encapsulated derived class (`output_soap_message`) is the answer to the query. This is especially important when the query consists of multiple rules, as in this case.

```
R10:
if   C@candidate_classifier(classifierID:CL)
then classify(!classifierID:CL)
```

```
R11:
if   CL@classify(!classifierID:CL1) and
     A@candidate_atts(classifierID=CL1,attribute:Att,value:Val) and
then classifyPair(!classifierID:CL1,attName:Att,attValue:Val)


R12:
if   CP@classifyPair(!classifierID:CL,attName:Att,attValue:Val)
then classifyVector(!classifierID:CL,classifyPair:list(CP))


R13:
if   CL@classify(!classifierID:CL1) and
     CV@classifyVector(!classifierID=CL1) and
then CL@classify(classifyVector:CV)


R14:
if   CL@classify(!classifierID:CL1) and
     C@candidate_classifier(classifierID=CL1,address:URL)
then xml_result(output_soap_message(!address:URL,classify:CL))
```

Further selection strategies can be defined by the user who must supply a complete set of X-DEVICE rules that start from the initial set of candidate classifiers, filter out some of them based on arbitrary criteria, and calculate the final set of candidate classifiers. The user should utilize the following classes as input to his/her query:

- `init_candidate(method,classifierID,address)`. Holds all the registered classifiers that share an input attribute with the incoming request. Attribute `method` stores the selection strategy of the request, `classifierID` points to the OID of the `classifier` object, and `address` holds the URL address of the corresponding *wsClassify* Web service. Attribute `method` has a value of `user` for user-defined selection strategy.
- `candidate_atts(classifierID,attribute,value)`. Holds all the input attributes and values of the input SOAP message that are shared with some registered classifier. Attribute `classifierID` points to the OID of the `classifier` object, `attribute` holds the name of the input attribute, and `value` holds the value of the input attribute.

The user-defined X-DEVICE query should return the set of selected classifiers as instances of the `candidate_classifier(classifierID, address)` derived class. Attribute `classifierID` points to the OID of the `classifier` object and `address` holds the URL address of the corresponding *wsClassify* Web service. However, the user can avoid using the above output class altogether if he/she makes some use of the pre-defined selection strategies, as the following example shows.

We assume that a user wants to define a new selection strategy, so that a classifier can be selected if at least one of the following conditions is true:

- The classification methodology is "Neural Network" and the input attribute "coverage" is at least 50%, or
- The classification methodology is "Decision Tree" and the input attribute "coverage" is at least 75%.

The following two rules use the `init_candidate` input class and copy an instance of this class to a new instance by changing the selection strategy from user (the name of the user-defined strategy) to the appropriate *At least N%* strategy, according to the classification method of the classifier. Notice that the user need not directly involve output class `candidate_classifier`, but only indirectly by re-feeding the `init_candidate` class.

```
R15:
if   I@init_candidate(method=user,classifierID:CL,address:URL) and
     CL@classifier(classificationMethod='Neural Network')
then init_candidate(method=50,classifierID:CL,address:URL)


R16:
if   I@init_candidate(method=user,classifierID:CL,address:URL) and
     CL@classifier(classificationMethod='Decision Tree')
then init_candidate(method=75,classifierID:CL,address:URL)
```

Finding Homogeneous Classifiers

One of the tasks of X-DEVICE is to provide to the *wsCluster* service the groups of homogeneous classifiers, i.e. the group of classifiers that have exactly the same input and classification attributes, using the following rules:

```
R17:
if   C@classifier(classificationAttribute:CA,inputAttribute:IA) and
     not G@group(classificationAttribute=CA,inputAttribute≡IA)
then group(classificationAttribute:CA,inputAttribute:IA)


R18:
if   G@group(class ificationAttribute:CA,inputAttribute:IA) and
     C@classifier(classificationAttribute=CA,inputAttributes≡IA)
then G@group(classifiers:list(C))
```

Rule R17 iterates over all classifiers and copies the classification and input attributes to an instance of group class. Notice that the group is only created if it does not already exist. In order to compare the multi-valued attribute `inputAttribute` we use the same-set operator (≡), since the order of elements in each list may vary. Rule R18 iterates all the "similar" classifiers, for each created group, and keeps their OID in the classifiers attribute of group, using the `list` aggregate function. Therefore, the group class has three attributes: `classificationAttribute`, `inputAttribute`, and `classifiers`; the two latter are multi-valued attributes.

Querying Registered Classifiers

The Web services of the portal might query X-DEVICE about the stored meta-data of the registered classifiers. The following is an example that generates the list of input attributes that are relevant for each classification attribute. This query is used by the portal to adjust dynamically the classification input page (see Figure 5).

```
R19:
if   C@classifier(attName.classificationAttribute:CA) and
     not Cl@corresponding_attributes(classificationAttribute=CA)
then corresponding_attributes(classificationAttribute:CA)


R20:
if   Cl@corresponding_attributes(classificationAttribute:CA) and
     C@classifier(attName.classificationAttribute=CA,attName.inputAttribute:IA)
then Cl@corresponding_attributes(inputAttribute:set(IA))
```

Rule R19 creates an instance of `corresponding_attributes` class for each distinct classification attribute and stores the name of the attribute in the attribute `classificationAttribute`. Rule R20 iterates over all distinct classification attributes, i.e. all instances of class `corresponding_attributes`, and then retrieves all the input attributes of all the classifiers that have the same classification attribute. These input attributes are stored in the multivalued attribute `inputAttribute`, using the `set` aggregate function. This function is similar to `list`, their only difference being that no duplicate values are stored inside the set, which is implemented as a Prolog list.

Finally, rule R21 creates a single instance of the class `all_classification_attributes` that holds a list (set) of all the distinct classification attributes. This query is also used by the portal to dynamically generate the values of the pull-down menu (Figure 5) of the classification data entry page.

```
R21:
if   C@classifier(attName.classificationAttribute:CA)
then all_classification_attributes(classAtt:set(CA))
```

CLUSTERING DISTRIBUTED CLASSIFIERS

The proposed approach of classifier clustering is based on the notion of classifier distance, its efficient calculation for all pairs of classifiers, and a clustering algorithm that takes as input the distances and outputs the clusters.

Classifier distance

We here define classifier distance as a measure of how different two classification models are and propose its empirical measurement based on the classifiers' predictions on instances with known classes of an independent data set. By independent, we mean a data set whose instances were not part of the classifiers' training set. This will ensure unbiased results, as the predictions of classifiers on their training data tend to be optimistic.

If all classifiers are distribution classifiers then we propose the use of distance measures like Euclidean Distance, Canberra Distance and Czekanowski Coefficient (Krzanowski, 1993). In this case, the distance of two classifiers is defined as the average distance of their output vectors with respect to all instances of the independent data set.

If all classifiers are hard classifiers, then some measures that can be used for calculating classifier (dis)similarity are Yule's Q statistic, the correlation coefficient, the disagreement measure and the double-fault measure (Shipp & Kuncheva, 2002).

If mixed types of classifiers are used, then one could adapt the distribution classifiers to the statistics for hard classifiers by using the class of maximum support, breaking ties arbitrarily. Another solution is to adapt the hard classifiers to the measures for distribution classifiers by giving a support of 1 to the predicted class and 0 to the rest. However, this will produce artificially increased distances between two classifiers of different type.

The proposed empirical evaluation of classifier distance exhibits the following beneficial properties:

- *Independence of the classifier type*. It is able to measure the distance of two classification models, whether they are decision trees, rules, neural networks, Bayesian classifiers, or other. This is useful in applications where different types of learning algorithms might be used at each distributed node.
- *Independence of the classifier opacity*. It is able to measure the distance of two classification models, even if they are black boxes, providing just an output with respect to an input. This is useful in applications where the models are coming from different organizations that might not want to share the details of their local models.

Distance calculation

WebDisC uses the disagreement measure for classifier distance because i) it is simple and fast to compute ii) it can be computed incrementally, iii) it gives a value that directly expresses the distance of two classifiers that can be used without any transformation for the clustering process, and iv) it can be used for mixed type of classifiers.

The following equation defines the disagreement measure for two hard classifiers, C_x and C_y and a data set D with M instances:

$$d_D\left(C_x, C_y\right) = \frac{\sum_{i=1}^{M} \delta_{x,y}^{i}}{M}$$

where $\delta_{x,y}^{i}$ equals 1 if classifiers C_x and C_y have different output on tuple i, and 0 otherwise.

Figure 12: Classifier distance calculation algorithm

```
Input:
   D: an array of M instances (union of N data samples)
   C: an array of N classifiers
Output:
   Dist: an array of $\frac{N(N-1)}{2}$ distances
Begin
   For i ← 1 to M
   begin
      // calculate the output of classifiers
      For x ← 1 to N
         O[x] ← C[x](D[i]);

      // update distances
      index ← 1;
      For x ← 1 to N-1
         For y ← x+1 to N
         begin
            If O[x] ≠ O[y] Then
               Dist[index] ← Dist[index]+1
            index ← index+1
         end
      end

   // normalize distances
   For index ← 1 To $\frac{N(N-1)}{2}$

         Dist[index] ← Dist[index]/M;
End
```

The algorithm in Figure 12 shows the actual distance calculation process. Let *D* be the union of the *N* data samples that the *wsCluster* Web service of the Portal gathers through the *wsGetData* Web service of the WebDisC Nodes. Let *DC* be the list of the *N* classifiers that the *wsCluster* Web service of the Portal gathers through the *wsGetClassifier* Web service of the WebDisC Nodes. For every instance of *D* we calculate the output of all classifiers and then we update the disagreements for each pair of classifiers, based on their output. In the end, the disagreements are normalized with respect to the number of instances that were used for calculating them. The final output of the algorithm is a vector Dist with the distance for each pair of classifiers based on the disagreement measure.

Clustering

Having calculated the pairwise distances of all distributed classifiers, we proceed by clustering them using hierarchical agglomerative clustering. We chose this clustering algorithm because it does not require from the user to specify the number of clusters which is completely unknown and it uses the pairwise distances of objects, which have already been computed for the distributed classifiers as explained in the previous section.

The clustering algorithm takes 3 inputs. The first input is the distance vector calculated by the algorithm in Figure 12. The second input is the method for evaluating inter-cluster distances. There are various methods that could be used here including single linkage, complete linkage, Ward's method and weighted average linkage (Kaufmann & Rousseeuw, 1990). WebDisC uses the weighted average linkage method. The third input is a cutoff value, that determines when the agglomeration of clusters will stop, in order to produce the final clustering result.

That final clustering result is stored within the X-DEVICE system along with the meta-data of the classifiers. This knowledge can be used to guide the selection of the distributed classifiers that will participate in a combination as explained in *The WebDisC System* section.

CONCLUSIONS AND FUTURE TRENDS

This chapter has presented the WebDisC system, an approach for the fusion of distributed classifiers based on Web services. Its main advantage over state-of-the-art systems for distributed classification is its versatility, interoperability and scalability which stems from the use of open and extensible standards based on XML for Web-based distributed computing. The use of the XML-based PMML language for classifier exchange further adds to the inter-operability of WebDisC. Clients can be easily developed in any programming language and operating system that is Web-aware.

From the point of view of classifier fusion methodology, WebDisC currently supports simple techniques that do not require re-training of a complex classification model. Yet, it provides the necessary infrastructure, through the *wsGetData* and *wsGetClassifier* Web services, for the implementation of any classifier fusion methodology that requires re-training. Adding methodologies demands the extension of the Portal's Java servlets, while the WebDisC Nodes do not require any modification at all. This shows that WebDisC is a highly scalable and extensible system for classifier fusion.

In addition, WebDisC implements a novel approach towards the detection of interesting similarities and differences among homogeneous classifiers. Clustering the distributed classifiers provides useful knowledge for guiding the selection of classifiers that will participate in the fusion process, thus enhancing the quality of the final classification results.

Furthermore, the X-DEVICE deductive object-oriented database system for XML data, provides powerful mechanisms for querying the registered classifiers. Heterogeneous and homogeneous classifiers can be easily selected and fused through the use of the standard classifier selection strategies. The users of the system can also fine-tune the selection of classifiers that will participate in the fusion process according to their requirements.

In the future, we intend to extend the system with more complex fusion methodologies that require re-training. We will also investigate the implementation of such methodologies under the constraint of avoiding moving raw data from the distributed databases (Tsoumakas & Vlahavas, 2002), in order to avoid increased network communication overhead.

We also intend to enrich the user-interface of WebDisC with a user-profiling system. Its purpose will be to keep the history of the user-defined classifier selection strategies for each different user of WebDisC. This way strategies that have been successfully used in the past by a user can be retrieved and re-used in the future.

Finally, we intend to address syntactic and semantic heterogeneity problems that arise from the possibly different schemas of the distributed databases by empowering WebDisC with domain-specific ontologies. This is an important future trend in Web information systems development, that is driven by the Semantic Web vision.

ACKNOWLEDGEMENTS

Dr. Nick Bassiliades was supported by a postdoctoral scholarship from the Greek Foundation of State Scholarshipes (F.S.S. – I.K.Y.)

REFERENCES

Bassiliades N. and Vlahavas I. (1997) Processing production rules in DEVICE, an active knowledge base system. Data and Knowledge Engineering, 24(2):117-155.

Bassiliades N., Vlahavas I. and Elmagarmid A.K. (2000) E-DEVICE: An extensible active knowledge base system with multiple rule type support. IEEE Transactions on Knowledge and Data Engineering, 12(5):824-844.

Bassiliades N., Vlahavas I. and Sampson D. (2003a) Using logic for querying XML data. In D. Taniar and W. Rahayu, editors, Web-Powered Databases, pages 1-35. Idea Group Publishing.

Bassiliades N., Vlahavas I., Elmagarmid A.K. and Houstis, E.N. (2003b) InterBase-KB: Integrating a knowledge base system with a multi-database system for data warehousing. IEEE Transactions on Knowledge and Data Engineering, 15(3), to appear.

Bellwood T., Clement L., Ehnebuske D., Hately A., Hondo M., Husband Y. L., Januszewski K., Lee S., McKee B., Munter J. and Von Riegen G. (2002) UDDI version 3.0. Retrieved May 15, 2003 from http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

Boag S., Chamberlin D., Fernandez M. F., Florescu D., Robie J. and Simeon J. (2002) XQuery 1.0: An XML query language. Retrieved May 15, 2003 from http://www.w3.org/TR/xquery/.

Botia J. A., Gomez-Skarmeta A. F., Velasco J. R. and Garijo. M. (2001). A proposal for meta-learning through a MAS (multi-agent system). In T. Wagner and O.F. Rana, editors, Infrastructure for Agents, LNAI1887, pages 226-233.

Box D., Ehnebuske D., Kakivaya G., Layman A., Mendelsohn N., Nielsen H. F., Thatte S. and Winer D. (2000). Simple Object Access Protocol (SOAP) version 1.1. Retrieved May 15, 2003 from http://www.w3.org/TR/SOAP/

Brittenham P. (2001). An overview of Web Services Inspection Language. Retrieved May 15, 2003 from http://www.ibm.com/developerworks/webservices/library/ws-wsilover.

Champion M., Ferris C., Newcomer E. and Orchard D. Web services architecture. Retrieved May 15, 2003 from http://www.w3.org/TR/ws-arch/.

Chinnici R., Gudgin M., Moreau J. and Weerawarana S. (2002). Web Services Description Language (WSDL) version 1.2 working draft. Retrieved May 15, 2003 from http://www.w3.org/TR/wsdll2/.

Data Mining Group Web site (2002). Retrieved May 15, 2003 from http://www.dmg.org/.

Diaz O. and Jaime. A. (1997) EXACT: An extensible approach to active object-oriented databases. VLDB Journal, 6(4):282-295.

Gray P.M.D., Kulkarni K.G. and Paton N.W. Object-Oriented Databases, A Semantic Data Model Approach. Prentice Hall, 1992.

Y. S. Huang and C. Y. Suen. A method for combining multiple experts for the recognition of unconstrained handwritten numericals. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17:90-93, 1995.

Kaufmann L. and Rousseeuw P. J. (1990) Finding Groups in Data: An Introduction to Cluster Analysis. Wiley Interscience.

Kittler J., Hatef M., Duin R. P. W. and Matas J. (1998). On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(3):226-238.

Krzanowski W. J. (1993). Priniciples of Multivariate Analysis: A user's perspective. Oxford Science Publications.

Kuncheva L.I., Kounchev R. K. and Zlatev R.Z. (1995). Aggregation of multiple classification decisions by fuzzy templates. In Proceedings of the Third European Congress on Intelligent Technologies and Soft Computing EUFIT'95, pages 1470-1474, Aachen, Germany.

Lam L. and Suen C. Y. (1995). Optimal combinations of pattern classifiers. Pattern Recognition Letters, 16:945-954.

Rogova G. (1994). Combining the results of several neural network classifiers. Neural Networks, 7:777-781.

Saitta L. (2000) Machine learning: A technological roadmap. Technical report, University of Amsterdam.

Shipp C. A. and Kuncheva L. I. (2002) Relationships between combination methods and measures of diversity in combining classifiers. Information Fusion, 3(2): 135-148.

Stolfo S. J., Prodromidis A. L., Tselepis S., Lee W. and Fan D. W. (1997). JAM: Java agents for meta-learning over distributed databases. In Proceedings of the AAAI-97 Workshop on AI Methods in Fraud and Risk Management.

Tahani H. and Keller J.M. (1990) Information fusion in computer vision using the fuzzy integral. IEEE Transaction on Systems, Man and Cybernetics, 20:733-741.

Tsoumakas G. and Vlahavas I. (2002) Effective stacking of distributed classifiers. In Proceedings of the 15th European Conference on Artificial Intelligence, pages 340-344.

WebDisC Web site (2003). Retrieved May 15, 2003 from http://lpis.csd.auth.gr/systems/webdisc.html.

Werges S. C. and Naylor D. L. (2002). Corba infrastructure for distributed learning and meta-learning. Knowledge-Based Systems, 15:139-144.

Wolpert D. (1992). Stacked generalization. Neural Networks, 5:241-259.

X-DEVICE Web site (2002). Retrieved May 15, 2003 from http://lpis.csd.auth.gr/systems/x-device.html.