Chapter 1

GENERIC HIERARCHICAL CLASSIFICATION USING THE SINGLE-LINK CLUSTERING

Willy Picard and Wojciech Cellary
Department of Information Technology
The Poznań University of Economics
Mansfelda 4, 60-854 Poznań, Poland
{picard, cellary}@kti.ae.poznan.pl

Abstract Up to date, research on automatic classification focused mainly on the efficiency of algorithms in regard to a given aspect of a dataset. The issue of classification of a given dataset in regard to various aspects is generally not addressed. In this chapter, a multi-facet hierarchical classification technique based on the single-link clustering is proposed. First, the single-link clustering is formally presented. Then, the concepts underlying the generic hierarchical classification technique are given. Next, analysis domains modeling a given facet of a dataset are described. A new language devoted to generate analysis domains is presented. Further, classification of analysis domains is discussed. Finally, examples of applications of the generic hierarchical classification are given.

Keywords: hierarchical classification, multi-facet analysis, ultra-metrics, single-link clustering, analysis domain language.

1. Introduction

With the growth of the amount of information available on the Internet, data classification is a necessity. Classification of information should be hierarchical in order to allow fast focusing on information classes of higher interest. Information classification problem is a subject of intensive research. Two classification methods that are the mostly used in practice are: Kohonen Self-Organizing Maps Algorithm and the Hierarchical Bayesian Clustering. The Kohonen Self-Organizing Maps

Algorithm [6, 7] is based on a unsupervised neural network mapping high dimensional input data onto a two-dimensional output space while preserving relations between the data items. The Hierarchical Bayesian Clustering [5] is based on Bayesian probability theory. It maximizes the probability that a cluster is included in another cluster. Both these methods are focused on the most efficient way of the dataset classification according to a given criterion. However, in practice, a dataset needs to be classified in many ways called below a "generic hierarchical classification". For example, scientific papers can be classified according to their domains, keywords, lengths, editors, relations with other papers, etc.

The Kohonen Self-Organizing Maps Algorithm is not suitable to the generic hierarchical classification, because the granularity of the data classification is determined by the predefined, fixed size of the output space.

The Hierarchical Bayesian Clustering is also not well suitable to generic hierarchical classification, because each new classification criterion would need the definition of a new probabilistic function for cluster aggregation. These probabilistic functions are difficult to built and even more difficult to intuitively understand for a human.

The approach to the generic hierarchical classification proposed in this chapter is based on the single-link clustering, which has strong mathematical basis presented in Section 1.2. The single-link clustering is based on the use of ultrametrics. It has been proven that the notions of ultrametric and hierarchical classification are equivalent. Thus, an ultrametric defines a hierarchical classification. Furthermore, an ultrametric may be derived from any metric. So, any metric defines a hierarchical classification.

The classes resulting of a classification based on a metric d are sets of elements that are similar according to the criterion defined by the distance d. The more similar two elements are, the less the distance between them is. The definition of a new classification criterion would imply the definition of the similarity between elements. Thus, a new classification criterion would imply the definition of a new metric. As the notion of metric is intuitive, new classification criteria may be easily defined.

In this chapter, first, the mathematical basis of the single-link clustering is presented. Then, the multi-facet analysis concept is introduced. Next, the data retrieval mechanism is described and formalized, as well

as the classification technique used. Finally, some examples illustrate the power of the multi-facet classification technique.

2. Single-link Clustering

The goal of any classification is to group items according to their proximity. The concept of promixity can be considered as the similarity between items. The more two items are similar, the closest their are.

2.1 Partition

Similarity between items is expressed by a mathematical concept of an *equivalence relation.*

DEFINITION 1.1 *An equivalence relation $\mathcal{R}$ on a set A is a relation that is reflexive ($\forall a \in A$, $a\mathcal{R}a$), symmetric ($\forall a, b \in A$, $a\mathcal{R}b \Leftrightarrow b\mathcal{R}a$), and transitive ($\forall a, b, c \in A, a\mathcal{R}b$ and $b\mathcal{R}c \Rightarrow a\mathcal{R}c$).*

Examples of equivalence relations are: "have the same car" or "live in the same country". An example of an equivalence relation on natural numbers is the "=" relation. The keyword of equivalence relations is the word "same".
Equivalence relations partition the universe into subsets (called *classes*):

DEFINITION 1.2 *A* partition *P of a set A is a collection of subsets $\{A_1, \ldots A_k\}$ such that any two of them are disjoint (for any $i \neq j$, $A_i \bigcap A_j = \emptyset$) and such that their union is A ($\bigcup_{i \in [1, \ldots n]} A_i = A$).*

Every element of A is a member of exactly one subset of the partition P. Assume that relation $\mathcal{R}$ is an equivalence relation on the set A. Let $[a]$ denote the set $\{b \in A \,|\, a\mathcal{R}b\}$, where $a \in A$.

LEMMA 1.3 *For $a \in A$, the sets $[a]$ constitute a partition of A.*

Proof. Let assume that $a, b \in A$ exist, where $[a] \neq [b]$ and $[a] \bigcap [b] \neq \emptyset$. Let c be any element of $[b] - [a]$. Let d be any element of $[a] \bigcap [b]$. First, $a\mathcal{R}d$, because $d \in [a]$. Second, $d\mathcal{R}b$, because $d \in [b]$, and $b\mathcal{R}c$, because $c \in [b]$ and $\mathcal{R}$ is symmetric. By transitivity, $d\mathcal{R}c$. Finally, by transitivity, $a\mathcal{R}c$, which means that $c \in [a]$. The last result is in contradiction with the fact that $c \in [b] - [a]$. Thus, for every $a, b \in A$, either $[a] = [b]$ or $[a] \cap [b] = \emptyset$, which means that, for $a \in A$, the sets $[a]$ constitute a partition of A. □

LEMMA 1.4 *Any partition $\{A_1, \ldots, A_k\}$ of A defines an equivalence relation by letting $a\mathcal{R}b$ iff a and b are members of the same A_i.*

Proof. Reflexivity. $\forall a \in A, \exists i \in [1, \ldots, k]$ such that $a \in A_i$. Clearly, a and a are members of the same A_i, which means that $a\mathcal{R}a$.

Symmetry. If $a\mathcal{R}b$, $\exists i \in [1, \ldots, k]$ such that $a \in A_i$ and $b \in A_i$. Also b and a are members of A_i, and therefore $b\mathcal{R}a$.

Transitivity. Assume that $a\mathcal{R}b$ and $b\mathcal{R}c$. Thus, there exist $i, j \in [1, \ldots, k]$ such that a and b are members of the same A_i and b and c are members of the same A_j. However, by definition of a partition, b cannot be a member of two different A_i and A_j. So $A_i = A_j$ and a and c are members of the same A_i, proving $a\mathcal{R}c$. □

By the proof of the two above lemmata, partitions and equivalence relations are exchangeable notions.

2.2 Indexed Hierarchy

Let assume that A is a finite set. Let $\mathcal{P}(A)$ denote the set of all subsets of A.

DEFINITION 1.5 *A hierarchy $\mathcal{H}$ on A is a subset of $\mathcal{P}(A)$ such that:*

$$A \in \mathcal{H}, \tag{1.1}$$

$$\forall a \in A, \quad \{a\} \in \mathcal{H}, \tag{1.2}$$

$$\forall (h_1, h_2) \in H^2, \quad \begin{cases} & h_1 \cap h_2 = \emptyset, \\ or & h_1 \subset h_2, \\ or & h_2 \subset h_1. \end{cases} \tag{1.3}$$

DEFINITION 1.6 *An indexed hierarchy on A is a couple $(\mathcal{H}, f)$, where $\mathcal{H}$ is a hierarchy and f is an application from $\mathcal{H}$ to $\mathbb{R}^+$ such that:*

$$\forall a \in A, \quad f(\{a\}) = 0, \tag{1.4}$$

$$\forall (h_1, h_2) \in H^2,\ h_1 \subset h_2, \quad h_1 \neq h_2, \ \Rightarrow f(h_1) < f(h_2). \tag{1.5}$$

2.3 Ultrametrics

DEFINITION 1.7 *An ultrametric on a set A is an application δ from $A \times A$ to $\mathbb{R}^+$ such that:*

$$\forall (a, b) \in A^2, \quad \delta(a, b) = 0 \Leftrightarrow a = b, \tag{1.6}$$

$$\forall (a, b) \in A^2, \quad \delta(a, b) = \delta(b, a), \tag{1.7}$$

$$\forall (a, b, c) \in A^3, \quad \delta(a, b) \leq \sup[\delta(a, c), \delta(b, c)] \tag{1.8}$$

LEMMA 1.8 *Consider δ an ultrametric on a set A. The relation $\mathcal{R}_{\delta_0}$ defined by*

$$\forall \delta_0 \in \mathbb{R}^+, a\mathcal{R}_{\delta_0}b \Leftrightarrow \delta(a, b) \leq \delta_0 \tag{1.9}$$

is an equivalence relation.

Proof. Reflexivity. $\forall a \in A, \delta(a,a) = 0$. Therefore $\delta(a,a) \leq \delta_0$, which means that $a\mathcal{R}a$.

Symmetry. If $a\mathcal{R}_{\delta_0} b$, then $\delta(a,b) \leq \delta_0$. By symmetry of ultrametrics, $\delta(a,b) = \delta(b,a)$. Therefore $\delta(b,a) \leq \delta_0$, which means that $b\mathcal{R}_{\delta_0} a$.

Transitivity. Assume that $a\mathcal{R}_{\delta_0} b$ and $b\mathcal{R}_{\delta_0} c$. Thus, $\delta(a,b) \leq \delta_0$ and $\delta(b,c) \leq \delta_0$. Because of equation 1.8, $\delta(a,c) \leq \sup[\delta(a,b), \delta(b,c)] \leq \sup[\delta_0, \delta_0] \leq \delta_0$,proving $a\mathcal{R}_{\delta_0} c$. □

As shown in Section 1.2.1, equivalence relations and partitions are exchangeable notions. Therefore, let $\mathcal{P}(\delta_0)$ denote the partition that consists of classes resulting from the equivalent relation $\mathcal{R}_{\delta_0}$.

LEMMA 1.9 *Let A be a finite set. If $\mathcal{H} = \bigcup_{\delta_0 \in \mathbb{R}^+} \mathcal{P}(\delta_0)$, then $\mathcal{H}$ is a hierarchy.*

Proof. Let us show that $A \in \mathcal{H}$. As A is finite, there exists a maximum distance between two its elements. The partition associated with the maximum distance is the set A.

It is obvious that $\forall a \in A, \{a\} \in \mathcal{H}$, because $\{a\} = \mathcal{P}(0)$.

Finally, for the condition 1.3, consider some arbitrary h_1 and $h_2 \in \mathcal{H}$. Then, $\exists(\delta_0, \delta_0^{'}) \in \mathbb{R}^+$ such that $h_1 \in \mathcal{P}(\delta_0), h_2 \in \mathcal{P}(\delta_0^{'})$. If $h_1 \cap h_2 = \emptyset$, the condition is observed. Otherwise, consider $a \in h_1 \cap h_2$. h_1 can be written as $h_1 = \{b \in A \,|\, \delta(a,b) \leq \delta_0\}$. Respectively, $h_2 = \{b \in A \,|\, \delta(a,b) \leq \delta_0^{'}\}$. If $\delta_0 \leq \delta_0^{'}$, then $h_1 \subset h_2$. Otherwise, $h_2 \subset h_1$, which concludes the proof. □

LEMMA 1.10 *Let f be an application from $\mathcal{H}$ to $\mathbb{R}^+$such that*

$$\forall h \in \mathcal{H}, f(h) = \min\{\delta_0 \,|\, h \in \mathcal{P}(\delta_0)\}.$$

Then, the couple $(\mathcal{H}, f)$ is an indexed hierarchy.

Proof. $\forall a \in A, \{a\} \in \mathcal{P}(0) \Rightarrow f(\{x\}) = 0$. Moreover, consider some arbitrary h_1 and $h_2 \in \mathcal{H}$, such that $h_1 \subset h_2$ and $h_1 \neq h_2$. Consider a being a member of h_1. Therefore:

$$\{\delta_0 \,|\, h_1 \in \mathcal{P}(\delta_0)\} \subset \{\delta_0 \,|\, h_2 \in \mathcal{P}(\delta_0)\},$$

and then

$$\min\{\delta_0 \,|\, h_1 \in \mathcal{P}(\delta_0)\} \leq \min\{\delta_0 \,|\, h_2 \in \mathcal{P}(\delta_0)\}.$$

So, $f(h_1) \leq f(h_2)$. Moreover, $h_1 \neq h_2$ and $h_1 \subset h_2$ implies that there exists an element b which is a member of h_2 but not a member of h_1. Consider $a \in h_1 \cap h_2$. Then,

$$f(h_1) < \delta(a, b) \leq f(h_2),$$

which means that $f(h_1) < f(h_2)$. □

LEMMA 1.11 *Consider an indexed hierarchy $(\mathcal{H}, f)$. An application δ from $A \times A$ to $\mathbb{R}^+$ is an ultrametric if*

$$\forall (a, b) \in A^2, \delta(a, b) = \min_{h \in \mathcal{H}} \left\{ f(h) \,|\, (a, b) \in h^2 \right\}.$$

Proof. Reflexivity. By definition, $\delta(a, a) = 0$, because $f(\{a\}) = 0$. If $\delta(a, b) = 0$, then

$$\exists h \in \mathcal{H} \,\text{such that}\, a \in h, b \in h, f(h) = 0.$$

If $h \neq (\{a\})$ (i.e. $a \neq b$), $f(h) > f(\{a\}) = 0$, which is impossible. Then $a = b$**.**

Symmetry. The demonstration is obvious.

Condition 1.8. Consider h_1 such that $\delta(a, b) = f(h_1)$, h_2 such that $\delta(a, c) = f(h_2)$, h_3 such that $\delta(b, c) = f(h_3)$. As $c \in h_2 \cap h_3$, then $h_2 \cap h_3 \neq \emptyset$. Because $\mathcal{H}$ is a hierarchy, let assume that $h_2 \subset h_3$ (if $h_3 \subset h2$, the proof is similar). Then,

$$f(h_2) \leq f(h_3). \tag{1.10}$$

Therefore, a, which is a member of h_2, is also a member of h_3 (b does so). As

$$f(h_1) = \min_{h \in \mathcal{H}} \{f(h) \,|\, (a, b) \in h^2\},$$

$h_1 \subset h_3$. Then,

$$f(h_1) \leq f(h_3). \tag{1.11}$$

Because of inequalities 1.10 and 1.11,

$$f(h_1) \leq \sup\left[f(h_2), f(h_3)\right],$$

which proves Condition 1.8. □

From Lemmata 1.10 and 1.11, the notions of an ultrametric and an indexed hierarchy are exchangeable, as illustrated in Figure 1.1. When δ_0 changes, different classes are created and the hierarchical aspect of the classification is visible in the embedment of classes.

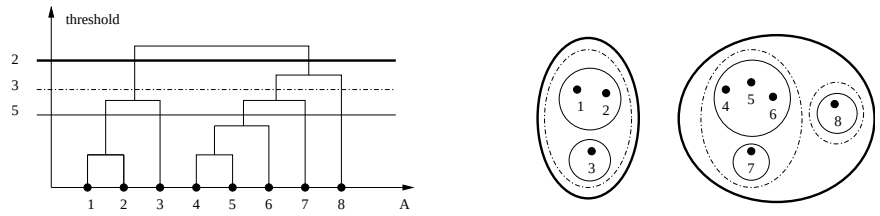

Figure 1.1. Equivalence between indexed hierarchy (on the left side) and ultrametric (on the right side, the classes for various values of δ_0)

2.4 Path Metrics

DEFINITION 1.12 *A* path *p between two elements a and b in a set A is a list of elements $p = (a_1, a_2, \dots a_n)$ such that:*

$$\begin{aligned} \forall i, \quad a_i &\in E, \\ a_1 &= a, \\ a_n &= b. \end{aligned}$$

DEFINITION 1.13 *Let d be a metric on A. A* step *$s_d(p)$ of the path p for the metric d is:*

$$s_d(p) = \sup_{i=1}^{n-1} d(a_i, a_{i+1}).$$

DEFINITION 1.14 *The* path metric *δ_d for metric d on a set A is an application from $A \times A$ to $\mathbb{R}^+$ such that:*

$$\delta_d(a, b) = \inf_{p \in \mathbb{P}(a,b)} s_d(p),$$

where $\mathbb{P}(a, b)$ is the set of all the paths from a to b in A.

LEMMA 1.15 *Each path metric is an ultrametric.*

Proof. Conditions 1.6 and 1.7 are obvious. Let prove condition 1.8.

$$\begin{aligned} \delta_d(a, b) &= \inf_{p \in \mathbb{P}(a,b)} s_d(p) \\ &\leq \inf_{p \in \mathbb{P}_c(a,b)} s_d(p), \end{aligned}$$

where $\mathbb{P}_c(a, b)$ is the set of all the paths from a to b in A containing c. By definition of $s_d(p)$,

$$\begin{aligned} \delta_d(a, b) &\leq \inf_{p_1 \in \mathbb{P}(a,c)} \inf_{p_2 \in \mathbb{P}(c,b)} \sup \left[s_d(p_1), s_d(p_2) \right] \\ \delta_d(a, b) &\leq \sup \left[\inf_{p_1 \in \mathbb{P}(a,c)} s_d(p_1), \inf_{p_2 \in \mathbb{P}(c,b)} s_d(p_2) \right] \\ \delta_d(a, b) &\leq \sup \left[\delta_d(a, c), \delta_d(c, b) \right], \end{aligned}$$

which proves Condition 1.8. □

A path metric may be derived from every metric. As every path metric is an ultrametric, and every ultrametric is equivalent to an indexed hierarchy,

Every metric defines an indexed hierarchy.

3. Concepts

3.1 Analysis of a Set of Data

Knowledge extraction is mainly basing on various analyses of a dataset. In many cases, users cannot conduct these analyses manually, because the amount of data to be analyzed is too high. Therefore, software tools need to be developed to provide users with synthetic views of a given dataset.

An analysis support system has to provide users with a possibility of various analyses of a dataset to well understand different aspects of the given dataset. For instance, a user may want to analyze the involvement of different scientists in the scientific community, or to analyze the correlation between authors and a given research topics.

To analyze a dataset , both the abstract objects to be analyzed and the analysis criteria must be defined.

3.2 Mapping Functions

Objects to be analyzed are generated by a mapping function f from space S to a set denoted D_f. S is the dataset space. Different mapping functions are used to analyze different aspects of the dataset.

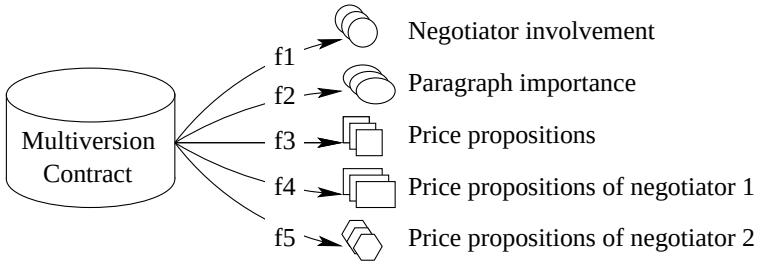

Figure 1.2. Mapping functions

To illustrate the use of various mapping functions to analyze different aspects of a dataset, consider a negotiation process in which various negotiators are trying to reach an agreement on a contract, each offer being considered as a new contract version. The various aspects of the dataset generated during the negotiation process can be analyzed with the help of various mapping functions as presented in Figure 1.2. Mapping function f_1 generates a set of objects modeling negotiator involvement. Mapping function f_2 generates a set of objects modeling paragraph importance. Mapping functions f_3, f_4, and f_5 generate sets of objects modeling price propositions for various subsets of the analyzed dataset.

3.3 Hierarchical Classification

In a highly concurrent environment, the result of an analysis should be a hierarchical classification [8]. Given a set of objects, a classification splits it into subsets of similar objects, denoted classes. Hierarchical classification provides users with a set of embedded classes. Users can then choose a granularity level (the number of classes) of the classification. For instance, the same set of authors will be split into few classes if a general involvement characteristics is required, or into many classes if detailed characteristics of authors involvement is required.

As the proposed solution is based on metrics, and ultrametrics, the criteria used to analyze set D_f are defined in a human understandable way, so that users may choose and define the criteria they want.

4. Analysis Domains

4.1 Domain Objects

Domain objects are used to model various facets of the dataset to be classified. Domain objects may for instance represent the editors of scientific papers, the relationships between news and press agencies, etc. As a consequence, domain objects must be flexible enough to represent various data types.

Each domain object is an element of an *analysis domain*. An analysis domain is a set of domain objects modeling a facet of a dataset. Formally, let D_f denote the analysis domain modeling facet of a dataset, denoted f.

A domain object DO_i is uniquely identified in a given set of domain objects by its identifier do_i. Formally,

$$\forall (DO_i, DO_j) \in D_f^2,\ do_i = do_j \Leftrightarrow DO_i = DO_j$$

$\forall(DO_i, DO_j) \in D_f \times D_{f'}$, $do_i = do_j$ and $DO_i \neq DO_j \Rightarrow D_f \neq D_{f'}$

A domain object DO_i consists of:

- a unique identifier, denoted do_i,
- a set of attributes, and
- a type.

An attribute is a pair $(name, value)$. Each attribute models a property of the domain object. To illustrate the use of attributes, let us assume that a book is modeled by a domain object denoted DO_{book}. The attributes of DO_{book} are pairs $('author','JohnSmith')$, $('editor','Kluwer')$, and $('title','the\ ACME\ prototype.')$.

An attribute name is a character string. A character string consists of one or many characters defined in the Unicode standard [9]. In case of Internet data classification, no limitations are assumed on the used languages. As a consequence, attribute names should not be limited to one or a few sets of languages and their characters. The use of Unicode allows for an international audience.

An attribute value is a domain object, because domain objects are able to model complex data types. A domain object may, for instance, model an editor, with attributes *name* and *address*. The value of the *editor* attribute in the former example may be such a domain object.

Also identifiers are domain objects in order to associate semantics with the identifiers. For instance, the identifier do_{book} may be a domain object modeling a "book identity". Identifier do_{book} may have the following attributes: *ISBN* (International Standard Book Number), and *checkDigit* (last digit of an ISBN used to check ISBN validity). The domain object DO_{book} is then identified by a domain object defining the ISBN identifier and its check digit.

Object domain types are Unicode character strings. Object domain types are used for two purposes. First, an object domain type associates some semantics with an object domain. In the former example, the type of DO_{book} may be *book* to indicate the meaning of the data DO_{book} models. Second, domain object types allows domain object structure to be defined. The structure of domain objects representing books may be defined as follow:

- identifier: type *extendedISBN*,
- attributes:

– author: type *Person*,
– editor: type *Editor*,
– title: type *String*.

Types of identifier and attribute values are corresponding to domain object types.

The following six primitive domain objects may be considered as the atomic data elements for domain object building:

- String,
- Integer,
- Long,
- Float.
- Double,
- Boolean.

Primitive domain objects do not refer to any other domain object type. Primitive objects have only one attribute denoted *value*. The value of this attribute depends on the domain object type. Primitive domain object values are presented in Table 1.1.

Table 1.1. Primitive domain objects

Type	*Description*	*Size/Format*
String	Character string	from 0 to $2^{31} - 1$ Unicode characters
Integer	Integer	32-bit two's complement
Long	Long integer	64-bit two's complement
Float	Single-precision floating point	32-bit IEEE 754 (defined in [3])
Double	Double-precision floating point	64-bit IEEE 754 (defined in [3])
Boolean	A boolean value (true or false)	true or false

4.2 Analysis Domain Definition

Domain objects are generated according to an *Analysis Domain Function (ADF)*. An ADF is a function whose image is an analysis domain. Formally,

$$f \text{ is an ADF } \Leftrightarrow \begin{cases} f \text{ is a function on analysis domains } D_{orig_i} \\ Im(f) = \{DO\}, \text{ where } DO \text{ are domain objects} \end{cases}$$

When two or more analysis domains exist for an ADF, the function is said to be *multi-variable*. A special analysis domain, denoted $\emptyset$, is defined by $card(\emptyset) = 0$. The existence of the analysis domain $\emptyset$ allows to distinguish *transformer* functions from *generator* functions.

DEFINITION 1.16 *An ADF f is a* generator *function iff only one origin domain of f exists that is $\emptyset$.*

A generator function creates an analysis domain without the need of pre-existing data in the form of an analysis domain. A generator function may for instance generate the number π, or retrieve data from a database.

DEFINITION 1.17 *An ADF f is a* transformer *function iff at least one origin domain of f is different from $\emptyset$.*

A transformer function transforms an existing analysis domain into another analysis domain. A transformer function may for instance transform an analysis domain modeling books into another analysis domain representing the number of books for each author.

ADF functions may be embedded according to the *composition law.* The composition law allows "pipelines" of functions to be defined, in which an analysis domain being the results of an ADF is an origin domain of another ADF.

DEFINITION 1.18 *The ADF composition law, denoted $\circ$, defines an ADF $f_\circ$ from ADF(s) f_i as follow:*

- *for single-variable functions, $f_\circ = f_1 \circ f_2 = f_1(f_2)$;*
- *for multi-variable functions, $f_\circ = f_1 \circ (f_2, \ldots, f_n) = f_1(f_2, \ldots, f_n)$,*

where f_1 is a transformator function, while f_2 and f_3 are either generator or transformator functions.

4.3 Analysis Domain Language

The *Analysis Domain Language (ADL)* is used to define ADFs. ADL is a dialect of XML — the eXtensible Markup Language. The eXtensible Markup Language [2] describes a class of data objects, called *XML documents,* and partially describes the behavior of computer programs that process them. XML is based on SGML – the Standard Generalized

Markup Language [4]. XML documents are conforming to SGML documents by construction. There are multiple reasons for using XML for ADL:

- it is an emerging standard developed by the WWW consortium;
- it is a general-purpose and extensible language;
- it allows defining language grammar that can be automatically validated by a parser;
- it is designed and optimized for parsing structured documents;
- the parsing software for XML is available;
- it can be easily integrated with other XML-based web standards;
- it allows defining human-readable data in a standardized way.

ADL is basing on four elements: *Metaobjects*, *ObjectSets, Tags,* and *Functions*. Metaobject correspond to domain objects. ObjectSets correspond to analysis domains. Tags are basic elements of processing. Functions correspond to ADFs.

4.3.1 Modules. ADL is structured in *modules*. A module groups metaobject definitions, definitions of functions generating these metaobjects and potentially implementation of needed features – as tags. A module may for instance define metaobjects modeling books and related-informations, functions for retrieval of book informations from a database and some new tags needed to access a database.

XML Namespaces [1] is used to avoid name collisions. Metaobject, function and tag names are universal, their scope extends beyond the module that contain them. Each module is responsible for associating itself with a URI. A module may use metaobjects, functions and tags defined in another module. A namespace referring the URI of the used module must be defined and associated with a prefix. An XML namespace must be associated to every used module.

An example of the use of XML Namespaces for modularization of ADL is presented in Figure 1.3. Metaobjects, functions and tags defined in module M_1 may be used in module M_2 as qualified names. M_2 may, for instance, associate the prefix *mOne* to module M_1. Function *fOne* defined in M_1 may be then used in M_2 as *mOne:fOne*.

A module is defined in an XML document. A module definition document contains:

- the name of the module,
- a URI defining the associated namespace,
- potentially a list of tag definition references,
- potentially a list of function definition references, and
- potentially a list of metaobject definition references.

An example of module definition document is given below:

```
<module name="testModule"
        uri="http://nessy.pl/adl/testing">
  <tags>
    <tag-decl name="if"
              definition="if.tdl"/>
    ...
  </tags>
  <functions>
    <function-decl name="BooksFromDB"
                   definition="retrieveBooks.fdl"/>
    ...
  </functions>
  <objects>
    <object-decl name="Book"
                 definition="Book.odl"/>
    ...
  </objects>
```

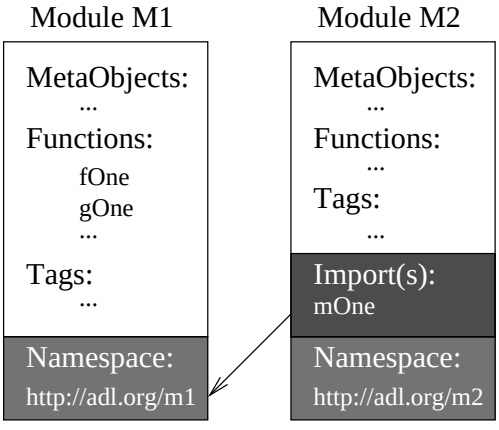

Figure 1.3. Modules in ADL.

```
</module>
```

In the module definition document given above, a module named `testModule` is defined. The URI `http://nessy.pl/adl/testing` is associated with this module. First, a tag named `if` is defined. Its definition can be found in the `if.tdl` tag definition document. Second, a function named `BooksFromDB` is defined. Its definition can be found in the `retrieveBooks.fdl` function definition document. Finally, a metaObject named `Book` is defined. Its definition can be found in the `Book.odl` metaobject definition document.

ADL provides a module — denoted the *core* module — which groups basic functionalities of ADL. The *core* module provides primitive metaObjects and fundamental tags for ADL. The namespace for the *core* module — the URI it is associated with — is *http://nessy.pl/adl/core*.

4.3.2 Expressions. In ADL, expressions are defined as ${*some Expression*}. An expression can contain:

- variable references,
- operators, and
- literals.

Variable reference
Variable references are done by names. If a variable *myVariable* has been defined, the expression ${*myVariable*} returns the variable *myVariable*. To test if *myVariable* is set, the expression ${*isEmptyObject␣myVariable* } may be used.

Operators
The following operators are defined:

- relational operators: ==, !=, ¡, ¿, ¡=, ¿=
- arithmetical operators: *, +, -, /, div, mod
- logical operators: ——, &&, !
- operator `empty` that checks if a metaobject is unset or if an objectSet contains some metaobject;
- operator "`.`" that retrieves metaobject attribute. For example, ${*myMetaObject.myAttribute*} retrieves the attribute *myAttribute* from metaobject *myMetaObject*;

- operator "[]" that retrieves metaobject from objectSet according to their IDs. For example, ${*myObjectSet*[*myMetaObjectID*]} retrieves the metaobject identified by *myMetaObjectID* from the objectSet *myObjectSet.*

Literals
The following literals are defined:

- logical: `true` or `false`,
- integers,
- floats,
- character strings surrounded by single or double quotes. The backslash character "\" is used to escape single and double quote characters, i.e. ""' is obtained by "\"'. The backslash character must be entered as "\\".
- Unset value: `null`.

4.3.3 MetaObjects. MetaObjects are defined in an XML document. A metaObject definition document contains:

- the name of the metaObject type,
- the name of the type of the metaObject identifier, a list of the attributes and their type if the metaObject is not a primitive,
- the type of the value if the metaObject is a primitive.

An example of metaObject definition document is given below:

```
<object-def type="Book">
  <id type="extendedISBN"/>
  <attributes>
    <attribute name="author" type="Person"/>
    <attribute name="title" type="String"/>
    <attribute name="editor" type="Editor"/>
  </attributes>
</object-def>
```

In the metaObject definition document presented above, a metaObject named `Book` is defined. It is identified by a metaObject whose type is `extendedISBN`. Three `Book` attributes are defined: `author` (of type `Person`), `title` (of type `String`) and `editor` (of type `Editor`). As attributes are defined, the metaObject `Book` is not a primitive metaObject.

An example of a primitive metaObject definition document is given below:

```
<object-def type="String">
  <value type="java.lang.String">
</object-def>
```

In the metaObject definition document presented above, the `String` primitive metaObject is associated with the `java.lang.String` class. No ID is defined as the `java.lang.String` is responsible for unique self-identification.

4.3.4 ObjectSets. An objectSet is a set of metaObjects. All metaObjects of a given objectSet have the same type. An objectSet may be empty, i.e. no metaObject is a member of the objectSet. The emptiness of an objectSet may be check with the `isEmptySet` operator. If objectSet *OS* is empty, *isEmptySet␣OS* is true. The emptiness of an objectSet may also be checked with the size operator. The size operator returns the size of an objectSet. The size of an objectSet is the number of metaObjects it contains. If the objectSet *OS* is empty, *OS.size* equals to 0.

The *core* module provides tags for basic operations on objectSets. Four operations are defined: objectSet creation (`declare` tag), metaObject addition to an objectSet (`add` tag), metaObject deletion from an objectSet (`remove` tag), and deletion of all metaObjects from an objectSet (`clear` tag). The `for-each` tag is an iterator on objectSets.

All ADF origin domains are objectSets. The image (in the mathematical sense) of all ADF is an objectSet. Figure 1.4 illustrates the relationship between ADF (functions) and objectSets.

4.3.5 Tags. Tags associate *processing entities* with XML tags. A processing entity is an independent software or a part of a software, such as a database access layer or a statistical library. The XML tags associated with processing entities can be used in function definitions. Tags are the only mechanism to extend ADL. When new features are needed, a new XML tag may be associated with a processing entity that implements the needed feature.

Two kinds of tags may are defined:

- empty tags, and
- non-empty tags.

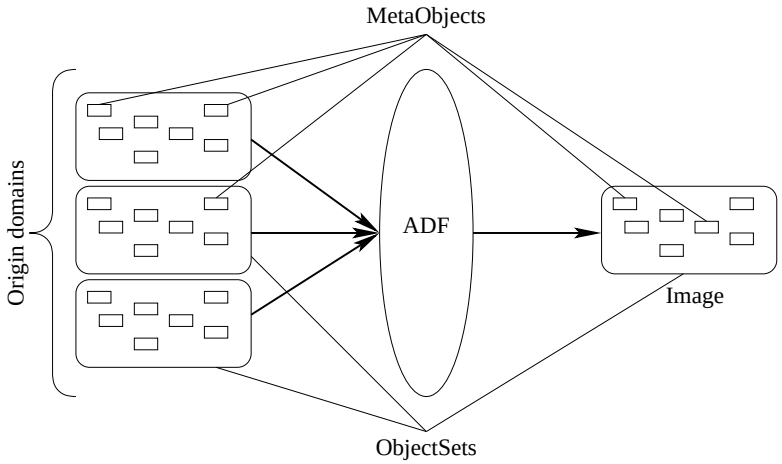

Figure 1.4. ObjectSets and ADF

An empty tag corresponds to an XML empty tag. An empty tag does not have any content. In a function declaration, an empty tag is called by the insertion of the associated empty XML tag.

A non-empty tag corresponds to a non-empty XML tag. A non-empty tag has content, with one or many children tags. In a function declaration, a non-empty tag is called by the associated non-empty XML tag.

Processing entities may be written in various programming languages. For the Java™ language, which has been chosen for the implementation of the ADL compiler, two interfaces have been defined: one for empty tags, and the other for non-empty tags.

Tags are defined in an XML document. A tag definition document contains:

- the name of a tag,
- the name of programming language a tag is implemented in,
- optionally – tag parameters specific to the chosen programming language.

An example of tag definition document is given below:

```
<tag-def name="SQLQuery" lang="java">
  <javaClass name="pl.nessy.db.SQLQuery">
    <params>
      <param name="connection"
             type="String"
             required="true"/>
```

```
      <param name="query"
             type="String"
             required="true"/>
    </params>
  </javaClass>
</tag-def>
```

In the tag definition document presented above, a tag named `SQLQuery` is defined. It is implemented in the Java™ programming language. All children (`connection` and `query`) elements are specific to tag implementation in Java™.

4.3.6 Functions. Functions are the core of ADL. An ADF is expressed in ADL as a function. Each function models a potential facet of a given dataset. A function processes zero, one or many objectSets. The result of the processing of a function is an objectSet. Generator ADFs, as defined in Section 1.4.3.7, are functions that do not process any objectSet. Transformers ADFs are functions that process at least one objectSet.

Functions are defined in an XML document. A function definition document contains:

- the name of a function,
- optionally – the names of modules the function uses,
- optionally – the objectSets to be processed and the type of metaObjects they contain,
- the name of the resulting objectSet and the type of metaObjects it contains,
- the processing actions to be performed.

The processing actions may be calls to tags and functions. Calls to tags are done by inserting the associated XML tags. To call functions, a special tag defined in the *core* module is applied.

An example of function definition document is given below:

```
1. <function-def name="BooksFromDB"
2.   xmlns:core="http://nessy.pl/adl/core"
3.   xmlns:col ="http://nessy.pl/adl/collections">
4.
5.   <param  name="isbnList"  type="ISBN"/>
6.   <param  name="bookDB"    type="BookFromDB"/>
```

```
 7.   <result name="books"     type="Book"/>
 8.
 9.   <processing>
10.     <declare name="tempBooks" type="Book"
11.              isASet="true"/>
12.     <core:for-each var="bookFromDB" items="bookDB">
13.       <core:declare name="localBook" type="Book"/>
14.       <core:set
15.         obj="localBookList"
16.         attribute="title"
17.         value="${bookFromDB.title}"/>
18.
19.      <col:ifContains
20.         col="isbnList"
21.         item="${bookDB.isbn}">
22.         <core:set
23.           obj="localBook"
24.           attribute="id"
25.           value="${bookDB.isbn}"/>
26.       </col:ifContains>
27.
28.       <add to-set="tempBooks" name="localBook"/>
29.     </core:for-each>
30.
31.     </core:execute name="col:deleteDoublons"
32.                    into="books">
33.        <core:param value="tempBooks"/>
34.     </core:execute>
35.   </processing>
36. </function-def>
```

In the function definition document presented above, a function named `BooksFromDB` is defined. It uses the `core` module (line 2) and a possible `col` module responsible for extended collection manipulation (3.). The function processes two objectSets `isbnList` and `bookDB` containing metaObjects of type `ISBN` and `BookFromDB` (lines 5 and 6), respectively. The resulting objectSet `books` contains metaObjects of type `Book` (line 7).

4.3.7 The *core* Module. The *core* module defines a set of primitive metaObjects and tags. Primitive objects are defined in Section 1.4.1. As presented in Table 1.2, tags in the *core* module are classified in five categories:

- variable declaration,
- metaObject attribute setting,
- control flow statements,
- function call, and
- objectSet manipulation operations.

Table 1.2. Tags defined in the *core* module

Tag category	*Tag(s) name*
Variable declaration	`declare`
MetaObject attribute setting	`set`
Control flow statement	`choose`, `if`
Function call	`execute`
ObjectSet manipulation	`add`, `for-each`, `remove`, `clear`

5. Classification of Domain Objects

5.1 Parametric Analysis

The goal of classification is to provide a synthetic view of an aspect of a given dataset. The choice of a facet corresponds to the choice of an ADF. The result of the execution of an ADF is an analysis domain, i.e. a set of domain objects. The ADF defines the facet of the dataset to be analyzed, generating domain objects modeling a given facet.

As domain objects may model complex views of a dataset, and the interests of a given user may be different from other users' interests, many analyses may be performed on the same domain objects. For this reason, the concept of parametric analysis is proposed.

DEFINITION 1.19 *An analysis is* parametric *if various criteria may be used to perform various analyses of a given analysis domain.*

A given analysis domain consists of a set of domain objects. All domain objects of a given analysis domain are of the same type. As a consequence, the type of an analysis domain may be defined as follows:

DEFINITION 1.20 *The* type *of an analysis domain* AD *is the type of domain objects of the analysis domain* AD*.*

The domain object type – and thus the analysis domain type – defines the attributes of all domain objects of this type. So, given an analysis

domain, the attributes of domain objects of the analysis domain are known.

Analysis criteria need the presence of some attributes. Attribute values are data to be analyzed. Attribute names are semantics associated with data to be analyzed. Therefore, every analysis criterion is associated with a given domain object type. As a consequence, the type of an analysis criterion may be defined as follows:

DEFINITION 1.21 *The* type *of an analysis criterion* AC *is the type of domain objects* the AC *is associated with.*

A given analysis domain may be associated only with the analysis criteria corresponding with the analysis domain type. A relationship many-to-many between analysis domains and analysis criteria exists. This relationship is a compatibility relationship.

DEFINITION 1.22 *An analysis domain* AD *and an analysis criterion* AC *are* compatible *iff the type of* AD *and the type of* AC *are the same*

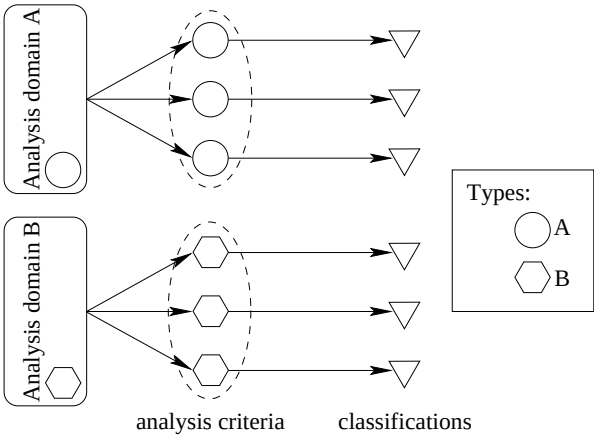

Figure 1.5. Parameterizable analysis

The concept of parametric analysis is illustrated in Figure 1.5. Two different analysis domains may be analyzed according to various criteria. Each criterion produces a classification. Each criterion is associated with an analysis domain type. In Figure 1.5, it is assumed that the domain analysis domain types of A and B are different. As a consequence, no criterion may be used for both analysis domain. The analysis domain A is *compatible* with the analysis criteria represented by a circle. The

analysis domain B is *compatible* with the analysis criteria represented by a hexagon. Analysis of A and B are parametric: various criteria may be used to perform various analyses of both A and B.

5.2 Analysis Criteria Definition

An analysis criterion is a metric on a given analysis domain. Formally:

DEFINITION 1.23 *A function AC is an* analysis criterion *iff*

$$\begin{cases} AC \text{ is a function from } AD^2 \text{ to } \mathbb{R}^+ \\ \forall (x,y) \in AD^2, \quad x = y \Leftrightarrow AC(x,y) = 0 \\ \forall (x,y) \in AD^2, \quad AC(x,y) = AC(y,x) \\ \forall (x,y,z) \in AD^3, \quad AC(x,y) \leq AC(x,z) + AC(y,z) \end{cases}$$

Analysis criteria are a subset of transformer functions (*Cf.* Section 1.4.2), so they are ADFs. They can, therefore, be defined in ADL.

An analysis criterion is an ADF with two analysis domain, each of them containing only one domain object. The resulting domain contains only one domain object modeling the concept of distance between two domain objects.

Formally, let AC denote an analysis criterion on an analysis domain AD. The type of AC, as well as AD, is denoted *typeAC*. Let do_1 and do_2 denote two domain objects of AD. Let $r \in \mathbb{R}^{+*}$ denote the distance between do_1 and do_2 according to AC. Let AD_r denote the resulting analysis domain.

For AC, two origin analysis domains AD_1 and AD_2 are defined by $AD_1 = \{do_1\}$ and $AD_2 = \{do_2\}$. The analysis criterion AC may be defined as follows:

```
<function-def name="AC"
  xmlns:core="http://nessy.pl/adl/core">

  <param  name="first"  type="typeAC"/>
  <param  name="second" type="typeAC"/>
  <result name="result" type="typeAC_distance"/>

  <processing>
  ...
  </processing>
</function>
```

AD_r contains only one domain object do_r. The type of do_r — which is also the type of AD_r — is *typeAC_*`distance`. Domain objects of this type may be defined as follows:

```
<object-def type="typeAC_distance">
  <id type="core:Integer"/>
  <attributes>
    <attribute name="first"    type="typeAC"/>
    <attribute name="second"   type="typeAC"/>
    <attribute name="distance" type="Float"/>
  </attributes>
</object-def>
```

Processing actions to be performed to calculate the distance between do_1 and do_2 are defined by calls to functions or tags, like for every ADF. The user defining new analysis criterion has to check whether the set of processing actions she/he uses to define the processing of the analysis criterion defines a metric. The ADF compiler does not perform any checking of the conditions given in Definition 1.23.

5.3 Classifications

Given an analysis domain and an analysis criterion that are compatible, a classification can be processed. A classification is the result of the analysis process. The structure of the classification is based on the inter-class distance. Classifications allow analysis domains to be partitioned at various level of granularity by the threshold operation.

5.3.1 Classification Structure. A classification is defined in terms of *classes* and *inter-classes distances*. A class may contain either other classes and an inter-class distance, or one or more metaObjects (with an inter-class distance always equal to 0). More formally, let C denote a classification on analysis domain DA. C is a set of classes denoted c_i. Classes are formally defined as follows:

$$\forall c_i \in C, \begin{cases} c_i = (\{c_j\}, d_i), \text{ where } \forall j,\ c_j \in C,\ d_i \in \mathbb{R}^{+*} \\ \text{or} \\ c_i = \{MO_j\}, \text{ where } \forall j,\ MO_j \in DA \end{cases}$$

$$\forall (c_i, c_j) \in C^2, c_i \subset c_j \text{ or } c_i \supset c_j \text{ or } c_i \bigcap c_j = \emptyset$$

Two kinds of classes exist. Some classes – denoted *atomic classes* – contains only metaObjects. Others – denoted *complex classes* – contains only classes and an inter-class distance.

The classification is an indexed hierarchy under the condition that an additional constraint is set on inter-class distances:

$$\forall (c_i, c_j) \in C^2 \text{ such as } c_i \neq c_j,\ c_i \subset c_j \Rightarrow d_i < d_j\ ,$$

where d_i (respectively d_j) is the inter-class distance associated with c_i (respectively c_j) and the inter-class distance of atomic classes equals 0.

Let a distance D on C be defined as follows:

- $c_i \subset c_j \Rightarrow D(c_i, c_j) = d_j$,
- if $c_i \bigcap c_j = \emptyset$, let $c_k \in C$ be such that $c_i \subset c_k$, $c_j \subset c_k$, and $\forall c_l \in C, c_l \subset c_k \Rightarrow \neg(c_i \subset c_l \text{ and } c_j \subset c_l)$. Then $D(c_i, c_j) = d_k$.

LEMMA 1.24 *In an indexed hierarchical classification, the inter-class distance d_i associated with class c_i observes $D(c_j, c_{j'}) \leq d_i$, where $c_j \subset c_i$ and $c_{j'} \subset c_i$.*

Proof. If $c_j \subset c_{j'}$, $D(c_j, c_{j'}) = d_{j'}$. As $c_j \subset c_i$, $d_j \leq d_j$. Therefore $D(c_j, c_{j'}) \leq d_i$.

If $c_{j'} \subset c_j$, a similar reasoning may be used.

If $c_j \bigcap c_{j'} = \emptyset$, there exists $c_k \in C$ such that $c_j \subset c_k$, $c_{j'} \subset c_k$, and $\forall c_l \in C, c_l \subset c_k \Rightarrow \neg(c_j \subset c_l \text{ and } c_{j'} \subset c_l)$. By definition, $D(c_j, c_{j'}) = d_k$. Or, as $c_i \in C$, $c_j \subset c_i$, and $c_{j'} \subset c_i$, by definition of c_k, $c_k \subset c_i$. Then $d_k \leq d_i$. As a conclusion, $D(c_j, c_{j'}) = d_k \leq d_i$. □

5.3.2 Classification Generation. Classification generation is an operation that, given an analysis domain and a compatible analysis criterion, generates a classification. Classification generation is based on ultrametric automatic hierarchical classification algorithm (cf. Section 1.2.3).

An analysis criterion is not an ultrametric. Analysis criteria only have to be metrics. However, a path metric – which is an ultrametric – can be derived from each metric (cf. Section 1.2.4). The path metric derivation is an operation whose computational complexity in terms of processing is very high. When the number of domain objects of an analysis domain

is n, a function processing all possible paths between two domain objects can be computed in $O(n!)$ time.
Proof. Consider an analysis domain AD such that $\text{card}(AD) = n + 2$. Let do_1 and do_2 denote two domain objects in AD. Let P_{do_1,do_2} denote the set of paths from do_1 to do_2. Then,

$$\text{card}(P_{do_1,do_2}) = n! \sum_{p=0}^{n-1} \frac{1}{(n-p)!}$$

As $\lim_{n\to\infty} \sum_{p=0}^{n-1} = e - 1$,

$$\lim_{n\to\infty} \text{card}(P_{do_1,do_2}) = \lim_{n\to\infty} n! \sum_{p=0}^{n-1} \frac{1}{(n-p)!} \simeq n!(e-1)$$

Therefore, a function processing all possible paths between two domain objects can be computed in $O(n!)$ time. □

We conclude that the path metric derivation operation cannot be performed directly because of the high complexity of the algorithm.

Another solution to derive ultrametric from metric is based on characteristics of ultrametrics:

LEMMA 1.25 *In an ultrametric space, all triangles are isosceles.*

Proof. Let δ denote an ultrametric on a space A. Let assume that three elements of A exist, a, b, and c, such that the triangle (a, b, c) is not isosceles. Assume that $\delta(a, b) < \delta(b, c) < \delta(a, c)$ — other cases are equivalent to this one by permutation of a, b, and c. Therefore, the condition $\delta(a, c) \leq \sup[\delta(a, b), \delta(b, c)]$ is not observed. As a conclusion, the hypothesis that a non isosceles triangle may exist is false. □

Using this characteristics of ultrametrics, we define the "isoscelization" operation. The "isoscelization" operation transforms every triangle in an isosceles triangle in which the three sides s_1, s_2, and s_3 observes the following: $\forall(i, j, k) \in [1, 2, 3]^3$, $s_i \leq \sup[s_j, s_k]$.

DEFINITION 1.26 *The "isoscelization" operation transforms a triangle* (a, b, c) *such that* $\begin{cases} \delta(a,b) = s_1 \\ \delta(a,c) = s_2 \\ \delta(b,c) = s_3 \\ s_1 \leq s_2 \leq s_3 \end{cases}$ *into a triangle* (a, b, c)*, where* $\begin{cases} \delta(a,b) = s'_1 = s_1 \\ \delta(a,c) = s'_2 = s_2 \\ \delta(b,c) = s'_3 = s_2 \end{cases}$.

The "isoscelization" operation is illustrated in Figure 1.6. The original triangle is the one on the left side. The transformed triangle is the one on the right side. The original triangle is a scalene triangle. After modification, the longest side – s_3 in the figure – is altered so that its length equals to the length of side s_2. Side s_2 is the side whose length is between s_1 – the smallest side – and s_3 – the longest side.

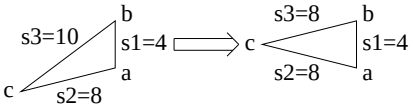

Figure 1.6. The "isoscelization" operation.

If the "isoscelization" operation is performed on all the triangles existing in a space A measurable with distance d, all the triangles will be isosceles. A new distance δ may be then defined on A as follows: the distance between two elements of A — denoted a_1 and a_2 — is the length of the segment a_1a_2 (in the space in which all triangles are isosceles).

It is worth to emphasize that the new distance δ obtained as explained above is an ultrametric. The proof is obvious as the constraints defining an ultrametrics, defined in Section 1.2.3, are observed. Constraints 1.6 and 1.7 come directly from properties of the metric d, and Constraint 1.8 is ensured by the "isoscelization" operation.

The complexity of the ultrametric processing based on the "isoscelization" operation is smaller than the processing based on path metrics. The complexity of the proposed algorithm is $O(n^3)$.

LEMMA 1.27 *Given a space A such that card$(A) = n$, the number of triangles existing in A equals $\frac{n(n-1)(n-2)}{6}$.*

Proof. A triangle consists of three points: a, b, and c. There are n possibilities for the choice of a. There are then $n-1$ possibilities for the choice of b. Finally, there are $n-2$ possibilities for the choice of c. So, there are $n(n-1)(n-2)$ triplets in A. Each triangle is counted six times with permutations: (a, b, c) is the same triangle as (a, c, b), etc. Therefore, the number of triangles in A equals $\frac{n(n-1)(n-2)}{6}$. □

The following algorithm may be used to perform the "isoscelization" of space A:

```
1. segments = orderedListOfAllSegments();
2. isoSegments = new List();
3. nonIsoSegments = orderedListOfAllSegments;
```

```
 4. while (nonIsoSegments.size() !=0 ) {
 5.   [a,b]=nonIsoSegments.firstElement();
 6.   for-each ( c in A, c <> a and c <> b ) {
 7.     isoscelization(a,b,c);
 8.   }
 9.   isoSegments.add([a,b]);
10.   nonIsoSegments.remove([a,b]);
11.   nonIsoSegments.sort();
12. }
```

The complexity of the presented algorithm is $O(n^3)$.

Proof. The number of segments in a space whose cardinal is n equals $\frac{n(n-1)}{2}$. For each segment, the loop defined between lines 6 and 8 is executed $n-2$ times. Therefore, the algorithm complexity is $\frac{n(n-1)}{2} \times (n-2) \simeq O(n^3)$. □

5.3.3 Threshold Operation. Lemma 1.24 allows to define a *threshold* operation on classifications. The threshold operation provides various partitions of the analyzed analysis domain according to a threshold.

Two additional concepts are required to define the threshold operation: the concept of *class contents* and the concept of *t-max class.*

DEFINITION 1.28 *The contents of a given class c_i of a classification C is a set of domain objects, denoted $Contents(c_i)$.*

- *For atomic classes, the class contents is the class itself, i.e. $Contents(c_i) = c_i = \{MO_j,\ where\ \forall j,\ MO_j \in DA\}$.*
- *For complex classes, the class contents is the union of contents of all embedded classes, i.e. $Contents(c_i) = \bigcup_j Contents(c_j)$ with the notations used in Section 1.5.3.1.*

The class contents operation is illustrated in Figure 1.7. The full classification C is presented in a). Domain objects are represented by circles. Five domain objects exist in C, denoted a, b, c, d, and e. Classes are represented by ellipses. Eight classes exist, denoted c_i, where $i \in [1, \ldots, 8]$. The "is-embedded-in" relationship is represented by lines between classes. Classes c_2 and c_3 are, for instance, embedded in c_1. For atomic classes (c_i, where $i \in [4, \ldots, 8]$), $Contents(c_i)$ is the set of domain objects contained in c_i. $Contents(c_4)$ is therefore $\{a\}$. When

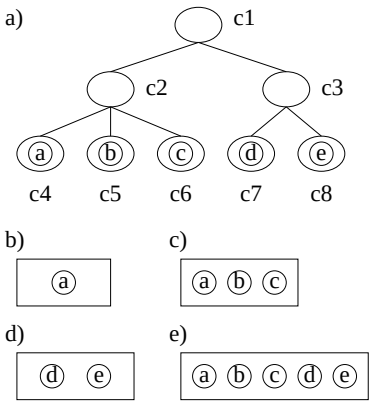

Figure 1.7. The *class contents* operation. a) the full classification C; b) $Contents(c_4)$; c) $Contents(c_2)$; d) $Contents(c_3)$; e) $Contents(c_1)$

a class is a complex class, the class contents is a set of object domains which is calculated recursively with the $Contents()$ function. Therefore, $Contents(c_2) = Contents(c_4) \cup Contents(c_5) \cup Contents(c_6) = \{a, b, c\}$. $Contents(c_3) = Contents(c_7) \cup Contents(c_8) = \{d, e\}$. Finally, $Contents(c_1) = Contents(c_2) \cup Contents(c_3) = \{a, b, c, d, e\}$.

DEFINITION 1.29 *Given $t \in \mathbb{R}^{+*}$, a class c_i is a t-max class iff*

$$d_i \leq t \text{ and } \neg(\exists c_j \text{ such that } c_i \subset c_j \text{ and } d_j \leq t)$$

In the classification presented in Figure 1.8, classes are represented by ellipses, distances associated with classes are given inside ellipses, "is-embedded-in" relationship is represented by lines between ellipses. In this classification example, c_2 is a 3-max class: the distance d_2 associated with c_2 observes $d_2 = 3 \leq t = 3$, and there is no class c_j such that $c_2 \subset c_j$ and $d_j \leq 3$ (the only class that contains c_2 is c_1, but $d_1 = 7$). Class c_3 is also a 3-max class: the distance d_3 associated with c_3 observes $d_3 = 2 \leq t = 3$, and there is no class c_j such that $c_3 \subset c_j$ and $d_j \leq 3$ (the only class that contains c_3 is c_1, but $d_1 = 7$).

DEFINITION 1.30 *Given a threshold t, the threshold operation T_t creates a partition P_t of a classification C. P_t is the set of contents of all t-max classes of C. Formally,*

$$P_t = \{Contents(c_i), \text{ where } c_i \text{ is a } t - max \text{ class}\}$$

Various threshold operation results are illustrated in Figure 1.9. When $t = 0$, the partition obtained by the threshold operation is a set of atomic

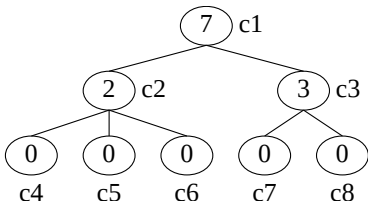

Figure 1.8. Classification example

classes. With the classification given in Figure 1.9 a), $P_0 = P_1$, because all 0-max classes are all 1-max classes. Similarly, $P_5 = P_7$, because all 5-max classes are 7-max classes.

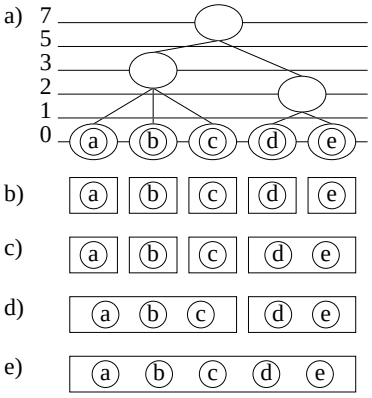

Figure 1.9. The threshold operation. a) Classification; b) P_1; c) P_2; d) P_3; e) P_7

The threshold provides the granularity of the obtained partition. The higher the threshold is, the lower the number of classes in the obtained partition is. In the context of knowledge extraction, this characteristics of the threshold operation is a key feature as it allows:

- various analysis levels; when the threshold is low, the generated partition consists of many classes, representing a fine-grained analysis. When the threshold is high, the generated partition consists of a few classes, representing a high-level analysis, giving an overview of the analyzed facet;
- fast focusing on details; starting from a high-level analysis, a user can select a few classes in the partition which are of special interest. These classes can further be analyzed in details by the application of a threshold operation with a lower threshold. The repetition of this technique allows to focus quickly on interesting details.

6. Applications

The universality of the ADL language – which is the basis of the proposed multi-facet analysis mechanism – allows for knowledge extraction in various areas and according to various criteria. Four knowledge extraction criteria may be distinguished: time-related, structural, contents-based, and combined.

Time-related knowledge extraction may be used to study the evolution of datasets and retrieve knowledge related to this evolution. Examples of time-related knowledge extraction are the study of a stock market evolution, or the analysis of a standardization process. Time-related knowledge extraction requires means of accessing past data, usually provided by a time-aware database or a content management system (CMS). ADL accessing these data source are used to model various aspects of a given time-related dataset. In the case of a stock market study, an ADL may generate metaObjects modeling the overall evolution of auctions, each metaObject providing the highest and the lowest values of the auction, as well as the dynamic of the auction. Auctions may then be classified according to various criteria, such as their dynamics or the mean value of their highest and lowest values. Therefore, knowledge concerning the dynamics of auctions in the stock market may be extracted from the generated hierarchical classification. Investors can then better understand the structure of the stock market, and, knowing the time-related behavior of various auctions, invest in a better – at least knowledge-based – way.

Structural knowledge extraction is based on the structural characteristics of data. In the case of the Web, structural knowledge extraction may base on the hypertext characteristics of data. Examples of structural knowledge extraction are the analysis of data source relevance on a given topic, or study of the organization of a web site. Structural knowledge extraction requires means of structure retrieval. ADL tags, specialized in structure retrieval, may be developed and reused. A tag retrieving a Web page and the hyperlinks it contains may be used by many ADFs. In the case of a study of data source relevance, an ADF may generate metaObjects modeling the the page structure, each metaObject providing the number of links in a given page, the list of addresses of these links, and the number of pages having at least one link to this page. Web pages may then be classified according to various criteria, such as their "reference" status (the highest number of links to a given page exists, the highest its "reference" status). Web pages may also

be classified according to the number of links to other pages, providing a classification of web pages to identify resources catalogs. Therefore, people interested in a given topic may have a faster access to important informations – "reference" pages – or may identify the pages containing meta-informations about the given topic. The structural knowledge extraction allows for identification of main elements in a given dataset.

Contents-based knowledge extraction may be used to study the organization of datasets in regard to its contents. Examples of contents-related knowledge extraction are the study of a set of documents to find main topics, or the classification of documents in regard to their relevance to a given topic. Content-based knowledge extraction requires means of filtering document contents, so that words with a low information quantity – such as "the", "a", or "is" – are not taken into account in the contents of documents. Tags filtering contents may be used by many ADFs which may model various aspects of a given set of documents. An ADF may for example generate metaObjects modeling the relevance of a document to the "physics" topics, each metaObject providing the number of occurrences of words "physics", "astrophysics", and "gravitation" in a given document and the identifier of the document. Documents may then be classified according to various criteria, such as their relevance to the gravitation topic or their relevance to the astrophysics topic. Another ADF may generate metaObjects modeling words and the number of their occurrence of words in the whole set of documents. Words may then be classified according to their frequency in the set of documents, allowing for topics extraction. Contents-based knowledge extraction allows for topics retrieval and identification of relevant documents to a given topic.

Combined knowledge extraction allows for dataset analysis in regard to mixed criteria: time-related, structural, and contents-based. An example of combined knowledge extraction is the study of centers of interest in the Java™ community in the last five years. A database can be used to stored web pages from Java-related web sites. An ADF may extract keywords from the database, another ADF may retrieve web pages associated with these keywords, and a structural analysis may be performed to retrieve relationships between keywords. Then, a last ADF may retrieve time-related data associated with the keywords. Finally, various classifications may be performed on the resulting metaObjects according to various criteria: center of interests can be classified according to the number of occurrence of a given word in the stored Web pages with a scaling factor implying a highest interest in "new" keywords; an-

other classification can stress on the structural knowledge retrieval with an influence of the number of occurrence of a given word, etc. Combined knowledge extraction provides complex analyses in which many factors are taken into account.

7. Conclusions

The generic hierarchical classification using the single-link clustering technique provides a solution to the problem of multi-facet analysis of a given dataset. The proposed technique may be the basis for knowledge extraction taking into account various aspects of a given dataset.

Two ideas that are the basis of the generic hierarchical classification technique are: first, the interests of various users concerning a given dataset can be different, second, expression of the subject and the criteria of the analyses must be easily understood be humans and extensible. As a consequence, the language ADL, used for definition of both the subjects of the analysis and the criteria, provides uniform means of expressing the various facets of a dataset to be analyzed.

Various application fields are possible for the generic hierarchical classification using the single-link clustering: time-related, structural, contents-based, and combined knowledge extraction. The combined knowledge extraction takes into account data evolution, structure of a dataset, and contents, allowing to extract information concerning structured, time-changing data.

The generic hierarchical classification technique opens new directions of research. An example is the use of software agents in electronic negotiations. Using the proposed analysis mechanism, advanced behavior models can be build for negotiating agents. Psychological and social models may base on data retrieved from the analysis of various facets of the negotiation process. An agent may for example have a "collaborative" behavior, i.e. may look for negotiators having similar proposals to build a group of negotiators in order to increase its weigh in the negotiation process.

References

[1] Bray T, Hollander D, and Layman A (1999), Namespaces in XML, W3C Recommendation, http://www.w3.org/TR/1999/REC-xml-names-19990114/

[2] Bray T, Paoli J, Sperberg-McQeen CM, and Maler E (2000), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, http://www.w3.org/TR/2000/REC-xml-20001006/

[3] IEEE (1987), IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std. 754-1985. *ACM SIGPLAN Notices 22*, 2.

[4] International Organization for Standardization (1986), Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). Tech. Rep. 8879:1986(E), ISO, Geneva, Swissland.

[5] Iwayama M and Tokunaga T (1995), Hierarchical Bayesian Clustering for Automatic Text Classification. In *Proceedings of IJCAI-95, 14th International Joint Conference on Artificial Intelligence*, pp. 1322–1327.

[6] Kohonen T (2001), *Self-Organizing Maps*, 3$^{\mathrm{rd}}$ ed. Information Sciences. Springer-Verlag, New York.

[7] Kohonen T (1990), The Self-Organizing Map. In *Proceedings of the IEEE*, vol. 78, pp. 1454–1480.

[8] Picard W (2001), Collaborative Document Edition in a Highly Concurrent Environment. In *First International Workshop on Web-Based Collaboration, at the 12th International Workshop on Database and Expert Systems Applications, DEXA 2001*, pp. 514–518.

[9] Unicode Consortium (2000), *The Unicode Standard, Version 3.0*. Addison-Wesley.