CHAPTER 8
MULTIMODAL FLOWER DOMAIN ONTOLOGY CREATION

The mere existence of natural living thing can be studied and analyzed efficiently only by ontology, where every existing resources are considered as an entity and the entities are grouped hierarchically via their relationships. The motivating concept of this work is to create low-level feature ontology for images, so that it would be easier to analyze and study the images automatically by a machine, thus a machine can visualize an image as human does. This would provide a way to develop a Semantic-based Image Retrieval System. As stated by Berners-Lee et al(2001) ontologies enable a better communication between humans and machines. Ontologies standardize and formalize the meaning of words through concepts. Ontology is represented by a Web Ontology Language called OWL. In OWL, each concept is represented as classes, sub-classes, and individual variables. The properties of each class can be represented by object properties and data properties. For these variables, logical relationship can also be specified using the basics of First Order Logics. In this work, the flower family domain of Asteroideae was considered. This family can also be called Daisy flower family as most of the genres belong to the category of daisies. There are nearly 1600 different genres in this flower family. Among them, 72 are distinct flowers and others are the shrubs type. For these genres, a domain-specific low-level feature ontology creation is elaborated in the following section.
Using the Asteroideae flower family domain knowledgeable information an Asteroideae flower image-feature domain (AFIF) ontology is created. The created ontology with respect to its syntactic image visual feature is discussed. The ontology creation is divided into two sections: top-level ontology’s entities creation and bottom level ontology’s entities creation. The creation of classes and sub-classes is said to be top-level ontology’s entity creation. Then the creation of data and object properties is said to be bottom level ontology’s entity creation. The top-level ontology entities creation mainly relies on the knowledge of Asteroideae flower family. Likewise, the creation of bottom level ontology entities mainly relies on the syntactic image visual features.

8.1 FUNDAMENTALS OF ONTOLOGY

In general, Ontology is a philosophical term, which is the study of the nature of human existence, knowledge, reasoning, mind, and language. After the invention of computing machines, the scientists started doing research on the development of intelligent agent which does the job as if human does. This domain of science is termed as artificial intelligence. In this domain to combine the human mind knowledge with machines, the ontology is introduced in computer science.

(Tom 2009) introduced the usage of ‘ontology’ in computing domain. According to the author, ontology was defined as:

“An ontology is a description (like a formal specification of a program) of the concepts and relationships that can formally exist for an agent or a community of agents.”
Therefore, the formal representation of concepts with respect to their relationship is said to be ontology. From the definition, the issues to be considered with respect to ontology are:

- The identification of accurate concepts
- The specification of relationship among the identified concepts.
- Representation of concepts and relationship
- Way to represent the constraints in relationship and concepts.

### 8.1.1 Types of Ontology

As per Zhao et al (2009), the ontology is broadly classified as static knowledge ontology and functional knowledge ontology as shown in Figure 8.1.

![Types of Ontology](image)

**Figure 8.1 Types of Ontology**

In static knowledge ontology, the identified concepts and relations are explicitly captured knowledge. Whereas the ontology under the functional ontology deliberately changes according to the knowledge acquisition.

The types under the static ontology are domain ontology, upper, top-level, or foundational ontology and generic or hybrid ontology. If ontology is designed with respect to specific domain such as medical diseases,
horticulture, heritage museum information etc., and such ontology can be labeled as Domain ontology.

As domain ontology is domain specific, to create ontology common to few domains, we can go on to build a top-level ontology. The upper ontology identifies the concepts, which are common across certain knowledge domain. The upper ontology supports wide semantic interoperability between large numbers of ontology. There are few famously available Ontologies under this category. They are UMBEL, BFO, GFO, IDEAS, WordNet, PROTON, ONSET etc.

The combination of domain specific and top-level ontology is said to be generic or hybrid ontology. The general representation of the concepts and their relationship without specifying the domain is said to be representational ontology.

In functional type, ontology is classified with respect to the task and method functionality of the ontology.

8.1.2 Component of Ontology

To provide semantic for knowledgeable information, the collection of information has to be represented as a set of assertions called ‘statement’ which consists of three part subject, object, and predicate which are also called triples. The subject and the object are the resource for which the semantics are to be built and the predicate is the relationship between the subject and object.

A language called as OWL (Web Ontology Language) which is the derivative of XML and RDFS represents the ontologies. The ontologies are usually stored as documents in web repositories. Such ontologies normally
consist of basic components such as optional header, annotations, class and property axioms, facts about individuals, and datatype definitions.

The ontologies optionally contain headers and the annotation, which describes the location and non-semantic description about the resource domain of the ontology. Therefore, the main building components of ontology are classes, individuals, and properties. The set of common resources are organized as classes. An individual is the instance of at least one of the class resources. The properties are of two types: data property and object property, which are used to provide semantic for the created classes and individuals.

8.1.2.1 Classes and individuals

The set of resources that shares the common characteristics are represented as ontology classes. The individuals are the resource that is a member of a class, which also represents an instance of that class. In ontology, the individuals can become members of class both directly by asserting the membership explicitly as well as implicitly by defining the membership condition for a class. Figure 8.2 shows a simple class hierarchy where the resource Seed_Bearing_Plant is a class and Angiosperms and Gymnosperms are individual instances of that class.

![Figure 8.2 Class and Individual relationship](image)
8.1.2.2 Properties

In ontology, properties are the resources that are used as predicate in statements that are used to describe the individuals. There are two main types of properties, namely, object properties that are used to provide semantic information between two different individuals and the data-type properties are used to provide literal values for the individual. The ontology supports most of all primitive data-type as of XML schema. In Figure 8.3, the individual Angiosperms and Gymnosperms have the data-type properties “types” which is of type string, used to provide information about different types of Angiosperms and Gymnosperms. Both the individuals are not equivalent which is specified by the object property “notSameAs”

![Diagram](Image)

Figure 8.3 Class and Property relationship

8.2 ONTOLOGY WITH RESPECT TO OWL

To represent explicit, formal conceptualizations of domain models ontology is required. To formulate ontology to a software language (Yu 2007) specified that it required a well-defined syntax, efficient reasoning support, a formal semantics, sufficient expressive power, and convenience of expression. The OWL is one such language. The OWL has the flavor of XML and RDFS thus it would be a well-defined language. However, the formal semantics are the concepts missed in XML kind of concept representation. A formal semantics describes the meaning of the domain knowledge precisely.
Thus, the described semantics applies to both human and machines. In OWL, the formal semantic are expressed by the mathematical logic. This logical expression will provide extraordinary reasoning power. For excellent reasoning support, the semantics are the prerequisite. The ontology reasoning support allows to check the consistency of the ontology and the knowledge, checks for unintended relationship between classes, and automatically classifies instances in classes.

8.2.1 Limitation of RDFS

- Local scope of properties: rdfs: range defines the range of a property; say eats, for all classes. Thus in RDF Schema we cannot declare range restrictions that apply to some classes only. For example, we cannot say that cows eat only plants, while other animals may eat meat, too.

- Disjointness of classes: Sometimes we wish to say that classes are disjoint. For example, male and female are disjoint. However, in RDF Schema we can only state subclass relationships, e.g., female is a subclass of person.

- Boolean combinations of classes: Sometimes we wish to build new classes by combining other classes using union, intersection, and complement. For example, we may wish to define the class person to be the disjoint union of the classes male and female. RDF Schema does not allow such definitions.

- Cardinality restrictions: Sometimes we wish to place restrictions on how many distinct values a property may or must take. For example, we would like to say that a person has exactly two
parents, or that at least one lecturer teaches a course. Again, such restrictions are impossible to express in RDF Schema.

- Special characteristics of properties: Sometimes it is useful to say that a property is transitive (like “greater than”), unique (like “is mother of”), or the inverse of another property (like “eats” and “is eaten by”).

### 8.2.2 Sample Representation of OWL

The OWL representation for the simple ontology shown in Figure 8.2 is clearly explained in the following section. The OWL is the combination of XML and RDF so it required both the header files so the optional header files section on the OWL are:

```xml
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">

<!-- OWL Header Example -->
    <owl:Ontology rdf:about="http://www.demo.com/plants">
        <dc:title>The Example Plant Ontology</dc:title>
    </owl:Ontology>

The Class definition: let us consider Seed_Bearing_Plant class as main class and Angiosperms and Gymnosperms as subclass. Let the Dasiy_flower and Pine be the individuals of the class Angiosperms and Gymnosperms respectively.
<!-- OWL Class Definition - Seed_Bearing_Plant -->
<owl:Class rdf:about="http://www.demo.com/plants# Seed_Bearing_Plant ">
  <rdfs:label>Seed_Bearing_Plant</rdfs:label>
  <rdfs:comment>The class of all seed bearing plant types.</rdfs:comment>
</owl:Class>

<!-- OWL Subclass Definition Angiosperms -->
<owl:Class rdf:about="http://www.demo.com/plants# Angiosperms ">
  <rdfs:subClassOf rdf:resource="http://www.demo.com/plants# Seed_Bearing_Plant "/>
  <rdfs:label>Angiosperms</rdfs:label>
  <rdfs:comment>Flowering plants, also known as angiosperms.</rdfs:comment>
</owl:Class>

<!-- OWL Subclass Definition Gymnosperms -->
<owl:Class rdf:about="http://www.demo.com/plants# Gymnosperms ">
  <rdfs:subClassOf rdf:resource="http://www.demo.com/plants# Seed_Bearing_Plant "/>
  <rdfs:label>Gymnosperms</rdfs:label>
  <rdfs:comment>Seed producing plants, also known as Gymnosperms</rdfs:comment>
</owl:Class>

<!-- Individual (Instance) Example RDF Statement -->
<rdf:Description rdf:about="http://www.demo.com/plants# Dasiy_flower ">
<!-- Dasiy_flower is a type (instance) of the Angiosperms classification -->

<rdf:type rdf:resource="http://www.demo.com/plants#Angiosperms"/>
</rdf:Description>
</rdf:RDF>

<!-- Individual (Instance) Example RDF Statement -->

<rdf:Description rdf:about="http://www.demo.com/plants#Pine">
  <!-- Pine is a type (instance) of the Gymnosperms classification -->
  <rdf:type rdf:resource="http://www.demo.com/plants#Gymnosperms"/>
</rdf:Description>
</rdf:RDF>

8.2.3 Adding Semantics to Ontology

Ontology is said to be effective only if it describes the context of the domain in it. To add semantics to the OWL properties, there are a number of ways. Certain concept involves describing a property in terms of another and the rest involve making a property a member of a class of properties, which provide the semantic meaning in OWL.

1. rdfs:subPropertyOf

Alike classes, properties can also be arranged hierarchically as justified by Yu et al (2007) using this syntax. In AFIF, ontology PC is one of the properties used to specify the prevalent color of the flower image. The color of the image can be specified by red, green, and blue color values. Therefore, PC would have the sub-property PCR, PCG, and PCB respectively.
2. Top and Bottom Properties

Both object and data-type properties have two corresponding properties: The properties are:

- owl:topObjectProperty - This property connects all possible pairs of individuals
- owl:topDataProperty - The role of this property is to connect all possible individuals with all literals.
- owl:bottomObjectProperty - Connects no pairs of individuals
- owl:bottomDataProperty - Does not connect any individual with a literal

3. Inverse Properties

The relationship between the subject and the object or from domain to range is asserted by properties. If the existence of the properties is implied in one direction, it will also exhibit certain relationship in other direction. Such properties are said to be inverse relationship. OWL uses the owl.inverseOf property to assert this property. Example of such combination is has_child => has_parent.

4. Disjoint Properties

To specify the disjoint between two properties the properties used are owl:propertyDisjointWith and owl:disjointPropertyWith. The disjoint properties mean that no two statements can exist where the subjects and objects of each statement are the same where the predicate is different. A
construct uses the class owl:AllDisjointProperties to specify pair-wise disjoint.

5. Set operator

The OWL provides three different set operation used to describe the membership of the classes. The operators are owl:intersectionOf, owl:unionOf and owl:complementOf. These operators are analogous to set theory. Each set operation established an equivalent class relationship with the class that it is describing.

So whatever set of instances the operation identifies, its result is implicitly the same as the extension of the described class. For owl:intersectionOf, all the individuals are the instances of all the classes. In owl:unionOf, the individuals are instances of at least one class. For owl:complementOf, individuals are not instances of any classes.

8.3 TOP LEVEL ONTOLOGY’S ENTITIES CREATION

Classes, Sub-classes, individuals, object properties, data properties and annotation properties are all the entities of the ontology.

The Asteroideae flower group was categorized as single flower and bunch flower with respect to the arrangement of flower’s corolla. Most of the flowers from this family would be of yellow, white, and purple color. So, these categories are subdivided with respect to the defined colors. The top level hierarchy of created ontology is shown in Figure 8.4
As per (OWL 2 Web Ontology Language 2012), the OWL can be represented as shown below. The `Single_flower` class declaration is shown:

```
Declaration (Class( :Single_flower)
SubClassOf(:Yellow_polypetalous :Single_flower)
SubClassOf(:White_polypetalous:Single_flower)
SubClassOf(:Purple_polypetalous:Single_flower)
SubClassOf(: Round_yellow_petalous:Single_flower)
```

The subclass assertion confirms that the individual instance of a particular subclass also belongs to its super class.
The OWL code snippet for the shown top level ontology creation is shown below:

```owl
<owl:Ontology rdf:about="Asteroideae flower domain ontology"/>

<!-- The Level two class declaration -->

<owl:Class rdf:ID="Single_flower"/>
<owl:Class rdf:ID="Bunch_flower"/>

<!-- The Level Three Class declaration -->
<!-- Subclasses of Single flower class -->

<owl:Class rdf:ID="Yellow_Poly-Petalous">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="#Single_flower"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="White_Poly-Petalous">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="#Single_flower"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Purple_Poly-Petalous">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="#Single_flower"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Yellow_Round_Petalous">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="#Single_flower"/>
  </rdfs:subClassOf>
</owl:Class>
```
<owl:Class rdf:ID="#Single_flower"/>
</rdfs:subClassOf>
</owl:Class>

<!—Subclasses of Bunch flower class -->

<owl:Class rdf:ID="Yellow-Petalous">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="#Bunch_flower"/>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="White-Petalous">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="#Bunch_flower"/>
  </rdfs:subClassOf>
</owl:Class>

Among each color dependent sub-classes, the specific flower genre are identified and made as sub-class for that group. The sample images of the flower group are shown in Figure 8.5. The flowers under yellow, white and purple polypetalous would be a single flower with many petal arrangements with respective colors. This flower family has more than 10 flower genres, which would be round in shape with small corolla and the flowers are grouped as round yellow petalous. Under bunch flowers, different groups of yellow and white tiny flowers are identified and grouped.
Figure 8.5 Asteroideae flower domain ontology Sample

The sub-class assertion of Yellow_polypetalous class which was again a sub-class of Single_flower class is shown below.

//Subclass assertion
SubClassOf(:Adenoglossa:Yellow_polypetalous)
SubClassOf(:Argyranthemum_maderence:Yellow_polypetalous)
SubClassOf(:Chrysanthemum_morifolium:Yellow_polypetalous)
SubClassOf(:Chrysanthemum_glandiforum:Yellow_polypetalous)
SubClassOf(:Cladanthus_arabicus:Yellow_polypetalous)
SubClassOf(:Coleostephus_myconis:Yellow_polypetalous)
SubClassOf(:Coleostephus_multicoulis:Yellow_polypetalous)
SubClassOf(:Cota_tincoria:Yellow_polypetalous)
SubClassOf(:Glossopappus:Yellow_polypetalous)
SubClassOf(:Heteranthemis:Yellow_polypetalous)
SubClassOf(:Lepidophorum:Yellow_polypetalous)
SubClassOf(:Prolongoa:Yellow_polypetalous)
SubClassOf(:Ursinia:Yellow_polypetalous)

The OWL code snippet for Round Yellow Petalous sub-class is shown below:

<owl:Class rdf:ID="Filifolium_sibiricum">
    <rdfs:subClassOf>
        <owl:Class rdf:ID="Yellow_round_Petalous"/>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="cotula_turbinata">
    <rdfs:subClassOf>
        <owl:Class rdf:about="#Yellow_round_Petalous"/>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="cotula_turbinata">
    <rdfs:subClassOf>
        <owl:Class rdf:about="#Yellow_round_Petalous"/>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Santolina">
    <rdfs:subClassOf>
        <owl:Class rdf:about="#Yellow_round_Petalous"/>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Gonospermum">
    <rdfs:subClassOf>
        <owl:Class rdf:about="#Yellow_round_Petalous"/>
    </rdfs:subClassOf>
</owl:Class>
After creating all the required classes and sub-classes, the instance, which is said to be the individual of the class, has to be created. This term, individual, is same as object creation in object-oriented programming artifact. Therefore, the individual of the class inherits all the properties of the parent classes.
Normally, an image retrieval system requires training sets, so the framework would try to match the given input image with respect to the trained image set. Likewise, in this ontology creation for each flower genre, there may be a requirement of two to three training images.

As in object-oriented programming paradigm individual in ontology is like instance of the class. In this AFIF ontology, the trained image data set for each flower genre are the individual data. For example, five images per genre were considered and named as per the genre name with numerals 1 to 5 in suffix such as Adenoglossa1. The named individual declaration for Adenoglossa class are:

```owl
//Declaration of individual
Declaration( NamedIndividual ( :Adenoglossa1)
Declaration( NamedIndividual ( :Adenoglossa2)

//Class assertion to specify the belonging class

ClassAssertion (:Adenoglossa :Adenoglossa1)
ClassAssertion (:Adenoglossa :Adenoglossa2)
```

So, for each flower class, say Pentzia, which is a sub-class of yellow_round_Petalous, there would have to be about three individuals, which can be named as Pentzia_1, Pentzia_2 and Pentzia_3. The OWL code snippet is shown:

```xml
<owl:Class rdf:ID="Pentzia">
<rdfs:subClassOf rdf:resource="#Yellow_round_Petalous" />
</owl:Class>

<Pentzia rdf:ID="Pentzia_1"/>
<Pentzia rdf:ID="Pentzia_2"/>
<Pentzia rdf:ID="Pentzia_3"/>
```
Once the basic entities like classes, sub-classes and individuals are created, the properties of this classes have to be specified, which will convert an ontology to a higher-level semantic ontology.

### 8.4 BOTTOM LEVEL ONTOLOGY’S ENTITIES CREATION

OWL has two main types: data properties and object properties referred to by Hebeler’s (2009) Semantic Web book. These properties are used to provide semantic for the created ontology. Object properties are used to provide the relation between two classes. While data properties are used to relate any literal data type to particular classes.

The syntactic features discussed in the above section would be the data property for each individual. Before finalizing the data property, the identified syntactic features have to be analyzed. To fulfill this ontology, three data properties have to be declared as shown in the listing below:

```plaintext
// Declaration of Syntactic feature Data property
Declaration ( DataProperty ( :PC))
Declaration ( DataProperty ( :BIP))
Declaration ( DataProperty ( :CGR))
```

where PC represents prevalent color, BIP means basic intrinsic pattern and CGR is contour gradient representation. The data-type of this property is “xsd:String” and it recommends values for PC = {Yellow, White, Purple} for BIP = {Smooth, Rough} and for CGR = {Round, Small, Large, Medium}.

So, if we consider Adenoglossa1 to be the individual instance of Adenoglossa class, then the data-property assertion is as shown below:
Yet to strengthen the ontology, identified normalized syntactic feature vectors are also included into the ontology by declaring the sub-property \{PCR,PCG,PCB\} for PC , \{BP1\} for BIP, and \{CGR5,CGR10\} for CGR respectively. Figure 8.6 shows the hierarchical semantics of the data property representation.

Three main data properties were declared, PC for prevalent color, BIP is used for basic intrinsic pattern representation, and CGR for contour gradient representation. As the prevalent color will have three distinct integer values for red, green, and blue, a separate data property is declared for RGB combination. Six different Eigen values of type double are calculated to identify the pattern of the given image. And finally, there would be 13 different gradient vector values which would be of type double. Below shown is the code snippet for the creation of PCR data property.

```
<owl:DatatypeProperty rdf:ID="PC"/>
<owl:DatatypeProperty rdf:ID="PCR">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Single_flower"/>
```

---

Figure 8.6 Data properties hierarchy

![Diagram of data property hierarchy]
In this ontological structure, same individual functionality was used to provide the semantic between two instance of same classes, PC value of same classes, and BIP value of same classes. To avoid the conflict of values between the classes, there is a need to make all the top-level classes disjointed with respect to each other, as shown.

```xml
<owl:Class rdf:ID="Single_flower">
  <owl:disjointWith rdf:resource="#Bunch_flower"/>
</owl:Class>
```

Thus, after creating the whole ontology structure, the corresponding data property values have to be specified. Let us say that the PC value for yellow colored images would be (255,255,128) with respect to RGB. This can be specified in each individual of the created image ontology as shown below:

```xml
<Pentzia rdf:ID="Pentzia_1">
  <PCR rdf:datatype="&xsd;integer">255</PCR>
  <PCG rdf:datatype="&xsd;integer">255</PCG>
  <PCB rdf:datatype="&xsd;integer">128</PCB>
</Pentzia>
```

<!-- the other data property values are -->

```xml
<Pentzia rdf:ID="Pentzia_1">
  <BIP1 rdf:datatype="&xsd;double">307398411</BIP1>
  <CGR5 rdf:datatype="&xsd;double">4.635</CGR5>
  <CGR10 rdf:datatype="&xsd;double">4.031</CGR10>
</Pentzia>
```
OWL has two types of properties, namely, data properties and object properties. These properties are used to provide semantic for the created class hierarchy. Object properties are used to emphasize the relation between the two classes. Data properties are used to relate any literal data type to a particular class.

For AFIF ontology, one object property is declared as “isSameAs”, which is used to connect all the values of 5 individuals. Thus, to consider all the five instances of a flower belonging to a single genre, this property is made to be symmetric, reflexive and functional.

```java
//Property Declaration
Declaration ( ObjectProperty ( :isSameAs))

//Object property assertion to relate two individual
ObjectPropertyAssertion ( :isSameAs : Adenoglossa1 : Adenoglossa2)

//The property axiom definition
SymmentricObjectProperty( :isSameAs)
FunctionalObjectProperty( :isSameAs)
IrreflexiveObjectProperty( :isSameAs)
```

8.5 CONSTRAINT BASED SYNTACTIC FEATURE PATTERN MINING

For a given data repository there would be too many patterns. To overcome this, Rule based constrain mining can be employed. In general for a given data repository $DR$ and constraint rule $RU$, by applying $RU$ on $DR$ it will identifies $PT$ pattern of class $C$. The initial step of any data mining technique is requirement analysis, data collection and data selection. For Asteroideae flower family, 500 flower images were collected. The flower images are analyzed and grouped as Yellow Polypetalous, White
Polypetalous, Purple Polypetalous and Yellow Petalous. This grouping is done by the domain knowledge of this flower family domain. These flowers are classified as ray flowers and disk flowers. The ray flowers of this flower family are round and yellow in color these are grouped as Yellow Petalous. The other groups are identified with respect to their corolla color.

From these images the syntactic low level features such as prevalent colour, basic intrinsic pattern and contour gradient are extracted. The prevalent color for each image would have values such as PCR, PCB and PCG which will ranges from 0 to 255. The basic intrinsic patterns (BIP) are used to identify the texture pattern of the image. The BIP would have the values ranges from 10 to 20. The shape related information of the image is calculated using contour gradient analysis (CGR). The values where ranging from 0.5 to 5. Constraint based pattern identification and clustering of this datum is elaborated in the forthcoming section.

### 8.5.1 Prevalent Color Pattern Mining

The basic dominant color of Asteroideae flowers are White, Purple and Yellow. If the RGB is [255,255,0] , the color is labeled as yellow; if it is [255,255,255], the color is white, and for purple the RGB value will be [128,128,255] . After extracting the prevalent color from the image, following rule constraint has to be checked to identify the pattern of data.

Let the binary decomposition of the data would be $PC = \{-1,0,1\}$ where, these value represent the color Purple, Yellow, and White respectively. Let $C_{PC} = 255$, the Rules for devising the color are elaborated in Equations (8.1 to 8.3)

\begin{align*}
\text{Rule 1} & : \{PC = 1, PCR < C_{PC} \land PCG < C_{PC} \land PCB = C_{PC}\} \\
\text{Rule 2} & : \{PC = 0, PCR = C_{PC} \land PCG = C_{PC} \land PCB < C_{PC}\} \\
\text{Rule 3} & : \{PC = -1, PCR = C_{PC} \land PCG = C_{PC} \land PCB = C_{PC}\}
\end{align*}
8.5.2 Basic Intrinsic Pattern Mining

With respect to the texture pattern the flowers are labeled as smooth or rough. The identified Eigen value-based basic intrinsic pattern is six to seven digit number, and natural log (ln) is applied to the number identified. If the value is above 17, the texture pattern is smooth whereas if it is below 15 - 16, the pattern is labeled as rough.

Let the binary decomposition of the intrinsic pattern be \( BP = \{0,1\} \) for smooth and rough texture pattern respectively. The threshold values are denoted as \( \alpha_{BIP} = 17 \) and \( \beta_{BIP} = 15 \). The constraint rules used to identify the pattern are shown in Equations (8.4 and 8.5)

\[
\text{Rule 4 : } \{BP = 1, \ BIP = \alpha_{BIP}, \land \ BIP < \alpha_{BIP}\} \tag{8.4}
\]

\[
\text{Rule 5 : } \{BP = 0, \ BIP = \beta_{BIP}, \land \ BIP < \alpha_{BIP}\} \tag{8.5}
\]

8.5.3 Contour Gradient Pattern Mining

The shapes of the flower are clustered as Round, Large, Medium, and Small. By analyzing the 13 different Texton image edges, the 5\textsuperscript{th} and 10\textsuperscript{th} filter PT5 and PT10 provide an effective pattern on the length and edge of the flower corolla. So, if those values lie in [4.0 - 4.9], then the flower is round in shape, if it lies in [2.0 - 2.9], the corolla is large; for a range of [1.0- 1.9] it is medium in shape, and if it is between [0.5 to 0.9], it is a tiny corolla. Let these values [4.0 - 4.9] be represented as SX and SM for small corolla flower maximum and minimum value. Likewise the other values can be denoted as MX,MM,LX,LM,RX and RM respectively. The binary decomposition of contour gradient pattern are \( CG = \{1,2,3,4\} \) which represent round, large, medium, and small shaped flower image. The constraint pattern rules are determined in Equation (8.6 – 8.9)
Rule 6: \{ \text{CGR} = 1, \text{CGR}5 = \text{RM}, \land \text{CGR}10 = \text{RM}, \land \text{CGR}5 < \text{RX}, \\
\land \text{CGR}10 < \text{RX} \} \quad (8.6)

Rule 7: \{ \text{CGR} = 3, \text{CGR}5 = \text{LM}, \land \text{CGR}10 = \text{LM}, \land \text{CGR}5 < \text{LX}, \\
\land \text{CGR}10 < \text{LX} \} \quad (8.7)

Rule 8: \{ \text{CGR} = 4, \text{CGR}5 = \text{MM}, \land \text{CGR}10 = \text{MM}, \land \text{CGR}5 < \text{MX}, \\
\land \text{CGR}10 < \text{MX} \} \quad (8.8)

Rule 9: \{ \text{CGR} = 2, \text{CGR}5 = \text{SM}, \land \text{CGR}10 = \text{SM}, \land \text{CGR}5 < \text{SX}, \\
\land \text{CGR}10 < \text{SX} \} \quad (8.9)

The extracted syntactic low level feature values for yellow polypetalous group are tabulated in Table 8.1. The cluster data labeled are specified in Table 8.2.

**Table 8.1 Syntactic feature analysis of Yellow Polypetalous**

<table>
<thead>
<tr>
<th>Yellow Poly-Petalous</th>
<th>PC</th>
<th>BIP</th>
<th>Ln(BIP)</th>
<th>CGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PCR</td>
<td>PCG</td>
<td>PCB</td>
<td>CGA5</td>
</tr>
<tr>
<td>Adenoglossa</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>69214778</td>
</tr>
<tr>
<td>Chrysanthemum glandiformum</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>67625076</td>
</tr>
<tr>
<td>Cladanthus arabicus</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>36441960</td>
</tr>
<tr>
<td>Coleostephus myconis</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>40718140</td>
</tr>
<tr>
<td>Coleostephus multicoulis</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>48291890</td>
</tr>
<tr>
<td>Cota tincoria</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>77946708</td>
</tr>
<tr>
<td>Glossopappus</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>35672406</td>
</tr>
<tr>
<td>Heteranthemis</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>53136974</td>
</tr>
<tr>
<td>Lepidophorum</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>35300960</td>
</tr>
<tr>
<td>Prolongoa</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>85023644</td>
</tr>
<tr>
<td>Ursinia</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>13253078</td>
</tr>
</tbody>
</table>
Table 8.2 Yellow Polypetalous Grouping as per the feature

<table>
<thead>
<tr>
<th>Name</th>
<th>Color</th>
<th>Texture</th>
<th>Edge-Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adenoglossa</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Medium</td>
</tr>
<tr>
<td>Argyranthemum maderence</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Medium</td>
</tr>
<tr>
<td>Chrysanthemum morifolium</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Large</td>
</tr>
<tr>
<td>Chrysanthemum glandiforum</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Large</td>
</tr>
<tr>
<td>Cladanthus arabicus</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Small</td>
</tr>
<tr>
<td>Coleostephus myconis</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Medium</td>
</tr>
<tr>
<td>Coleostephus multicoulis</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Medium</td>
</tr>
<tr>
<td>Cota tincoria</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Small</td>
</tr>
<tr>
<td>Glossopappus</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Medium</td>
</tr>
<tr>
<td>Heteranthemis</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Large</td>
</tr>
<tr>
<td>Lepidophorum</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Medium</td>
</tr>
<tr>
<td>Prolongoa</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Small</td>
</tr>
<tr>
<td>Ursinia</td>
<td>Yellow</td>
<td>Smooth</td>
<td>Large</td>
</tr>
</tbody>
</table>

8.6 BRIDGING THE SEMANTIC GAP USING ONTOLOGY

The threshold values obtained from the constraint based mining technique are used to provide semantics to the created ontology. The AFIF ontology’s data property values are provided from the details gathered from the constraint mining technique. The identified feature vectors are grouped as specified in Figure 8.7.
Figure 8.7 Syntactic Feature representation

So, let us consider Adenoglossa1 to be the individual instance of Adenoglossa class, and then the Annotation Assertion is as shown below:

// AnnotationAssertion
AnnotationAssertion( rdfs:label :PC: Adenoglossa1 "Yellow." )
AnnotationAssertion( rdfs:label :BIP: Adenoglossa1 "Smooth " )
AnnotationAssertion( rdfs:label :CGR: Adenoglossa1 " Medium " )

There are more than 70 classes which will have 5 individual instances each. To avoid the conflict between feature vectors values, a class axiom has to be defined to make the classes as disjoint. To make the Single_flower class and Bunch_flower class as disjoint declare the following axioms:

DisjointClasses ( :Single_flower :Bunch_flower)
Thus by declaring this way, the sub-classes of these classes will be also disjoint with each other. In order to make the sub-classes of a class to be disjoint, the following axiom can be used:

\[
\text{DisjointUnion( } \text{: Single\_flower } : \text{Yellow\_polypetalous } \\
: \text{White\_polypetalous } : \text{Purple\_polypetalous } : \text{Round\_yellow\_petalous} \text{ )}
\]

So, with this defined OWL entities and axioms, a perfect Asteroideae flower Image feature ontology was constructed. To query this constructed ontology, an OWL API-based java framework was used. This ontology can be used as the backend database for a semantic-based search engine.

### 8.7 SEMANTIC SEARCH - SYSTEM OVERVIEW

To implement the search engine concept as a service, it required a layered architecture diagram. Figure 8.8 shows one such diagram for the overall system overview. Up to this end the concentration of ontology which would narrow the semantic gap was elaborated. Now how to utilize the constructed ontology is shown in the Figure 8.8. The layered architecture of the web service is slightly modified with respect to the Semantic web services as specified by (Design and Implementation Guidelines for Web Clients 2003). The top-most layer is the application layer. In the application layer the user interface GUI is implemented. The semantic web services can be built either by using the Java OWL API package or the Visual studio based programmed library files. Here for simplicity we had used Visual Studio based framework. The front design would be the interface which would accept either user keyword or image as input and list out all the relevant information about the given keyword.
In business logic layer, the searching service routine programs are implemented on the server. The searching service routine would be of Rule extraction and feature matching.

![System Architecture of Proposed Semantic Search Engine](image)

**Figure 8.8 System Architecture of Proposed Semantic Search Engine.**

The next layer would be Ontology layer, where the created AFIF Ontology will be placed as a back-end process. In the created ontology the resources of each Flower class are all linked to the web pages which have the detailed information about the particular image. The last layer is the Assets layer where the flower related resources would be placed.

### 8.8 SUMMARY

In this chapter the basics of OWL based ontology creation was provided. The basic entity used to create the AFIF ontology was elaborated with necessary code snippet. To provide visual semantic about the image feature into the AFIF ontology the usage of constraint based pattern mining was specified. Using the pattern mining concept the threshold to index the
image with respect to the PC, BIP, and CGR based visual features were identified. Thus indexing and providing these information on ontology will reduce the semantic gap. The requirement to build the semantic web based image retrieval system was also elaborated in this chapter. Thus the created AFIF ontology can be used as the back end system for the semantic web based image retrieval system.