2004

Soft semantic for web information

Wafa Khamisy
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Khamisy, Wafa, "Soft semantic for web information" (2004). *Master's Theses*. 2585.
DOI: <https://doi.org/10.31979/etd.zant-39a2>
https://scholarworks.sjsu.edu/etd_theses/2585

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

SOFT SEMANTIC FOR WEB INFORMATION

A Thesis

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

by

Wafa Khamisy

May 2004

UMI Number: 1420467

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 1420467

Copyright 2004 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© 2004

Wafa Khamisy

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

A handwritten signature in black ink, appearing to read "Chris Tseng", written over a horizontal line.

Dr. Chris Tseng, San Jose State University, San Jose, California.

A handwritten signature in black ink, appearing to read "T.Y. Lin", written over a horizontal line.

Dr. T.Y. Lin, San Jose State University, San Jose, California.

A handwritten signature in black ink, appearing to read "Jon Pearce", written over a horizontal line.

Dr. Jon Pearce, San Jose State University, San Jose, California.

APPROVED FOR THE UNIVERSITY

A handwritten signature in black ink, appearing to read "Pamela C Stark", written over a horizontal line.

ABSTACT

SOFT SEMANTIC FOR WEB INFORMATION

By Wafa Khamisy

The Semantic Web, introduced by Tim Berners-Lee, uses Resource Description Framework (RDF) to add structure and meaning to Web applications. Existing RDF languages deal with hard semantics that are suitable for describing and processing crisp data. These languages cannot deal with soft semantics that include perception descriptions inherent in natural languages. On the other hand, Fuzzy systems, based on Fuzzy logic, can deal with perception-based data and process it to conclusion. This report describes how to integrate Fuzzy systems in The Semantic Web using existing methodologies. In addition, the report proposes a scheme to describe Fuzzy systems and their components in RDF and Extensible Markup Language (XML) format.

ACKNOWLEDGMENT

I would like to acknowledge Dr. Tseng for his technical guidance and support throughout my thesis work. I would also like to acknowledge my husband for supporting me throughout my graduate studies. He was always there to support and encourage me, and I appreciate his support greatly.

TABLE OF CONTENTS

1.	<i>Problem Statement</i>	1
2.	<i>Introduction</i>	1
3.	<i>The Semantic Web Overview</i>	2
3.1	<i>The Semantic Web Architecture</i>	3
3.1.1	<i>Unicode and Universe Resource Identifier (URI)</i>	3
3.1.2	<i>Extensible Markup Language (XML), XML Schema, and Namespace</i>	4
3.1.2.1	<i>XML Schema and Namespaces Overview</i>	6
3.1.3	<i>Resource Description Framework (RDF) and RDF Schema</i>	7
3.1.3.1	<i>RDF Syntax</i>	9
3.1.3.2	<i>Introduction to RDF Schema</i>	12
3.1.3.3	<i>RDF Example</i>	13
3.1.4	<i>Ontology Vocabulary</i>	15
3.1.5	<i>Logic, Proof, and Trust</i>	16
4.	<i>Fuzzy Logic</i>	16
4.1	<i>Fuzzy Set versus Crisp Set</i>	17
4.2	<i>Linguistic Variable</i>	19
4.3	<i>Membership Functions</i>	19
4.4	<i>Fuzzy Inference System</i>	21
4.5	<i>A “Tipper” Fuzzy System</i>	26
5.	<i>Incorporating Fuzzy Logic into The Semantic Web – a New Approach to Process Perception Based Information</i>	30
5.1	<i>Fuzzy System Representation in The Semantic Web</i>	31
5.2	<i>Fuzzy System Components in Semantic Web Applications</i>	36
5.3	<i>Implementing the Proposed Architecture- an Example</i>	42
6.	<i>Conclusion</i>	53
	<i>References</i>	54

<i>Appendices</i>	<i>56</i>
<i>Appendix A – Generalized Constrained Language (GCL) and The Semantic Web</i>	<i>56</i>
<i>GCL Overview.... ..</i>	<i>56</i>
<i>GCL and The Semantic Web</i>	<i>57</i>
<i>Appendix B – “Operator” XML Schema.....</i>	<i>70</i>
<i>Appendix C – “Pre_defined_function” XML Schema.....</i>	<i>76</i>

LIST OF FIGURES

<i>Figure 1 - The Semantic Web Architecture.....</i>	<i>3</i>
<i>Figure 2 - RDF Labeled Graph Example 1</i>	<i>9</i>
<i>Figure 3 - Class Hierarchy for RDF Schema.....</i>	<i>13</i>
<i>Figure 4 - RDF Labeled Graph Example 2</i>	<i>13</i>
<i>Figure 5 - Membership Function for "Old" Age.....</i>	<i>17</i>
<i>Figure 6 - Crisp Membership for "Old" Age.....</i>	<i>18</i>
<i>Figure 7 - A Triangular Shaped Curve.....</i>	<i>20</i>
<i>Figure 8 - A Trapezoidal Shaped Curve.....</i>	<i>20</i>
<i>Figure 9 - A Gaussian Shaped Curve</i>	<i>21</i>
<i>Figure 10 - Fuzzy Inference System Components.....</i>	<i>22</i>
<i>Figure 11 - The Min-Max Operation.....</i>	<i>24</i>
<i>Figure 12 - The Maximum Height Method of Defuzzification.....</i>	<i>26</i>
<i>Figure 13 - Service Membership Functions.....</i>	<i>27</i>
<i>Figure 14 - The Food Membership Functions.....</i>	<i>28</i>
<i>Figure 15 - The Tip Membership Functions.....</i>	<i>29</i>
<i>Figure 16 - Min-Max Operation for the Tipper Example.....</i>	<i>29</i>
<i>Figure 17 - Fuzzy System Description in RDF Labeled Graph.....</i>	<i>34</i>
<i>Figure 18 - RDF for Fuzzy IF-THEN Rule (1)</i>	<i>34</i>
<i>Figure 19 - RDF for Fuzzy IF-THEN Rule (2)</i>	<i>35</i>
<i>Figure 20 – "FUNCTION" XML Schema.....</i>	<i>35</i>
<i>Figure 21 – "User-defined-function" XML Schema</i>	<i>36</i>

<i>Figure 22 - Fuzzy System Components in The Semantic Web – A High Level Description</i>	38
<i>Figure 23 – Fuzzy System Components in Semantic Web – Detailed Interactions</i>	40
<i>Figure 24 – School Search Fuzzy System Architecture</i>	43
<i>Figure 25 - School User Interface</i>	44
<i>Figure 26 - School Class Hierarchy</i>	45
<i>Figure 27 – RDF/XML Sample Code for School Hierarchy</i>	45
<i>Figure 28 – RDF/XML Data Sample</i>	47
<i>Figure 29 - Property Hierarchy for Schools</i>	47
<i>Figure 30 - RDF Schema Sample Code for School Properties</i>	48
<i>Figure 31 - "API_HIGH" Function in XML</i>	49
<i>Figure 32 - School System Possible Output</i>	53

LIST OF TABLES

<i>Table 1: Membership Functions for School System.....</i>	<i>50</i>
<i>Table 2: IF-THEN Rule Set for the School System.....</i>	<i>50</i>
<i>Table 3: Rule Evaluations for Mission School.....</i>	<i>51</i>

1. Problem Statement

Existing Semantic Web deals with hard semantics that are suitable for describing crisp data (measurement-based knowledge), such as “Temperature is 80 Celsius.” It does not have the capability to process soft semantics that is able to deal with perception-based data, such as “Temperature is high.”

This report proposes a solution to show how perception-based information can be processed in Semantic Web applications. It proposes a scheme to integrate Fuzzy logic in The Semantic Web.

2. Introduction

The Web, a publishing medium and a repository for storing, sharing, and displaying text and images, uses Hyper Text Markup Language (HTML) to represent and process information on the Web. Later, Extensible Markup Language (XML) was added to provide structure and semantics to data. Lately, The Semantic Web was introduced to add meaning to the Web knowledge base.

The Semantic Web uses XML, Resource Description Framework (RDF), and ontologies to represent and process Web information. Once information is in RDF form, stored in RDF/XML (implementation of RDF through XML) document, taxonomies (class hierarchies), relations, and functions are defined, then a new inference can be derived from existing data.

“ The semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation “ – Tim Berners-Lee, 2001.

This report provides a solution for the problem of processing perception-based information in The Semantic Web, and integrating Fuzzy systems with each other and with other Semantic Web applications that use Fuzzy logic in their solutions. It shows how to integrate the notion of Fuzzy logic and Fuzzy sets with The Semantic Web by using existing methodologies. Specifically, it shows how to extend the meaning of resource properties to Fuzzy sets, and use RDF Schema and XML Schema to define a new vocabulary that describes their membership functions, and other related taxonomies. It also defines how to represent perception-based Web information in RDF format so that further inferences and analysis of data can be accomplished.

This report is organized as follows. Section 3 gives an overview of The Semantic Web, XML, XML Schema, RDF, and RDF Schema. Section 4 provides an overview of the Fuzzy Logic and Fuzzy Inference System. Section 5 gives a detailed description of how Fuzzy Logic can be part of The Semantic Web. Finally, a conclusion is provided in section 6.

3. The Semantic Web Overview

The Semantic Web is a scheme that was introduced by Tim Berners-Lee to extend the current Web from documents linked to each other, into a Web that recognizes the meaning of information in these documents. The following section discusses The Semantic Web architecture and its seven layers.

3.1 The Semantic Web Architecture

The Semantic Web consists of the following seven layers (Figure 1):

1. *Unicode and Universe Resource Identifier (URI)*
2. *Extensible Markup Language (XML), XML Schema, and Namespace*
3. *Resource Description Framework (RDF) and RDF Schema*
4. *Ontology Vocabulary*
5. *Logic*
6. *Proof*
7. *Trust*

Figure 1- The Semantic Web Architecture

3.1.1 Unicode and Universe Resource Identifier (URI)

A Unicode, a universal character code, is a 16 bits character set standard, designed and maintained by non-profit consortium Unicode Inc. It is universal and unique that covers all major modern written languages. Each character has exactly one

encoding and has to be represented by a fixed width in bits. A universal character encoding is required to produce software that can be localized for any language, or that can process and communicate data in any language.

Every Web resource (i.e., web page, image, video clips, services, and files) has a URI that makes the resource available under a variety of naming schemes and access methods. A URI is an identifier that consists of a sequence of characters.

3.1.2 Extensible Markup Language (XML), XML Schema, and Namespace

This section provides a brief description of the main features of XML and XML Schema.

XML defines text formats for structuring data. It uses tags to delimit pieces of data, and leaves the interpretation of the data to the application that reads it. It is used to describe, exchange, and store data. XML is platform independent, supports internationalizations, localization, and name space mechanism. It can also be considered as an ordered labeled tree, where the leaf nodes contain actual data, comments, document type definitions, and annotations. The element node has a type and a set of attributes.

A "Valid" XML document is a "Well Formed" (must start with XML declaration, must have one root element, element tags must be properly nested, start and end tags matches, etc.) XML document, that conforms to schema requirements. For example, the following shows a note from Mat to Dave stored in an XML format:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note xmlns="http://www.example.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com note.xsd">
```



```
<to> Dave </to>  
<from> Mat </from>  
<heading> Lunch </heading>  
<body> Would you like to eat lunch at 13:30 on Wednesday ? </body>  
</note>
```

The first line of the above example is the XML declaration. It defines the XML version and the character encoding used in the example. This XML example conforms to the 1.0 specification of XML and uses the ISO-8859-1 character set. The next three lines describe the root element, “*note*,” the default namespace declaration, the XML Schema Instance namespace, and the “*SchemaLocation*” attribute. The default namespace declaration states that all the elements used in the above XML example are declared in “*www.example.com*” namespace. The XML Schema Instance namespace states that the XML Schema Instances are declared in “*http://www.w3.org/2001/XMLSchema-instance*” namespace. The “*schemaLocation*” attribute has two values. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace (i.e., *note.xsd*). A schema is the grammar that defines the syntax of an XML document, such as Data Type Definition (DTD) and XML Schema (W3C, 1999). The next four lines describe all the child elements of the root: “*to*,” “*from*,” “*heading*,” and “*body*.” Finally, the last line defines the end of the root element, *</note>*.

An XML document does not contain any presentation information. This information is contained in the Extensible Style Sheet Language (XSL) file, which is based on the XSL Transformations (XSLT) language for transforming XML document into other formats. Separating the document's content and the styling information allows

the same document to be displayed with different formats depending on the application reading it (W3C, 1999).

3.1.2.1 XML Schema and Namespaces Overview

An XML Schema Definition (XSD) (W3C, 1999) is used to describe the structure of an XML document. It is written in XML, which makes it extensible. It supports namespaces and integrity constraints. Namespaces define a way to distinguish between duplicate element types and attributes that may occur from different schemas. A Namespace is identified by a unique URI that consists of two parts: the name of its XML namespace and its local name (W3C, 1999).

XML Schema can constrain the order and the number of children elements in mixed content model. It can specify the default values and range limits for data types.

<SCHEMA> tag is the root element of every XML Schema. XML Schema defines two types of elements: a simple type and a complex type. A simple type element can only contain text of type integer, string, date, and more. It has no attributes. The complex type element can contain other elements and attributes. For example, the following is an XML Schema that defines the elements of the above (note) XML example:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.com"
            xmlns="http://www.example.com">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

The first line of the above XML Schema is the XML declaration. The next three lines describe the root element “*schema*,” the XML Schema elements and data types (string, boolean, integer, and etc.) namespace, “*http://www.w3.org/2001/XMLSchema*,” the target namespace, “*http://www.example.com*,” and the default namespace, “*www.example.com*.” The next ten lines describe the root element, “*note*,” as a complex data type with four child elements: “*to*,” “*from*,” “*heading*,” and “*body*” of type “*string*.” Finally, the last line defines the end of the root element, *</schema>*.

3.1.3 Resource Description Framework (RDF) and RDF Schema

Web Information is designed for human consumption and it is a machine-readable and not machine-understandable. It is extremely challenging to manage Web information because of its ever-growing size. It is almost impossible to manage it manually. One of the solutions is to use metadata to describe the data on the Web. Web metadata can be defined as “data describing Web resources.” The **purpose** of RDF (Resource Description Framework) is to give a standard way of specifying data “about” something. It defines a mechanism for describing Web resources. RDF provides a common framework for representing metadata about resources that can be identified on the World Wide Web. Basically, it is a format for providing information in a machine-readable form, using

Universe Resource Identifiers (URI), and text strings as terms to identify resources and properties. This format makes it possible to exchange information between applications.

The fundamental design pattern of RDF is to structure your XML data as resource /property/value triples:

- **Resource:** Any object that is described by RDF expressions is called a resource. A resource can be a Web page, part of a Web page, a collection of Web pages, or an object that is not accessible via the Web, such as a book.
- **Property:** A property is an attribute or a relation used to describe a resource.
- **Value:** The value of a property can be a literal or a resource.
- **Statement:** A statement (triple) consists of three individual parts, namely: the subject, the predicate, and the object. The triple describes a relationship between a subject and an object by a property described in the predicate. The object of a statement can be another resource or a literal (a simple string or a data type defined by an XML Schema). For example, consider the following sentence:

Mary is the writer of book X.

This sentence has the following RDF syntax:

```
<rdf:RDF>
  <rdf:Description about = "http://www.example.org/books/X">
    <Writer> Mary </Writer>
  </rdf:Description>
</rdf:RDF>
```

Where the subject (<http://www.example.org/books/X>) has a property (Writer) that its value is the object (Mary).

A RDF URI can refer to any identifiable thing, including things that are not network-accessible or physically exist. Also, each RDF statement is a resource that can be used in other statements.

A set of RDF triples (statements) can be represented as a directed labeled graph (nodes and arcs diagram), where the nodes (drawn as ovals) represent resources and arcs represent named properties. Nodes that represent string literals will be drawn as rectangles. The direction of the arc is important. The arc always starts at the subject and points to the object of the statement. The above example can be diagrammed as illustrated in Figure 2.

Figure 2 - RDF Labeled Graph Example 1

3.1.3.1 *RDF Syntax*

The following section provides an overview of the main RDF elements used in this report:

- **rdf:RDF**

The *RDF* element is a wrapper that marks the boundaries in an XML document, where the content is explicitly intended to be an RDF data model instance.

- **rdf:Description**

The *Description* element is the place to hold the identification (using the *about* or *ID* attributes) of the resource being described. Typically, there will be more than one statement made about a resource. The *Description* element provides a way to give the resource a name just once for several statements. Both `rdf:about` and `rdf:ID` are attributes that identify a resource by RDF URI reference. The `rdf:about` provides absolute URI reference, while `rdf:ID` provides a relative one.

- **`rdf:resource`**

All things described by RDF are called resource, and are instances of the class `rdfs:Resource`. This is the class of everything. Resources are divided into groups of classes. The members of the class are known as instances of the class. Classes are themselves resources.

- **`rdf:subject`**

The *subject* element identifies the resource being described.

- **`rdf:predicate`**

The *predicate* element identifies the original subject's property.

- **`rdf:object`**

The *object* element identifies the subject's property value.

- **`rdf:Statement`**

A *Statement* is a resource reifying a triple. It has at least three properties: `rdf:subject`, `rdf:predicate`, and `rdf:object`.

- **`rdf:Property`**

A *RDF Property* is a relation between *subject* resource and *object* resource.

- **rdf:type**

RDF *type* is an instance of `rdf:Property` that is used to state that a resource is an instance of a class.

- **RDF Containers**

- **rdfs:Containers**

It is the super class of the RDF container classes, i.e., `rdf:Bag`, `rdf:Seq`, and `rdf:Alt`.

- **rdf:Bag**

The RDF *Bag* is a container that groups together relevant resources.

The order of resources is not important, and it can contain duplicate resources. The individual resources are individually numbered with `rdf:_1`, `rdf:_2`, `rdf:_3`, and etc.

- **rdf:Seq**

The RDF *Seq* is a container that groups together resources. The order of resources is significant. The individual resources are individually numbered with `rdf:_1`, `rdf:_2`, `rdf:_3`, and etc.

- **rdf:Alt**

The RDF *Alt* is a container that groups together resources, which can be arbitrary, substituted one for another. The order of resources is not important, and it can contain duplicate resources. The individual resources are individually numbered with `rdf:_1`, `rdf:_2`, and etc.

3.1.3.2 Introduction to RDF Schema

One of the problems of RDF is that you cannot specify the predicates' data types and how they relate to each other. RDF Schema, RDF's vocabulary description language, consists of some basic classes and properties that can be extended to fit almost any domain. RDF Schema is used to define class hierarchy (using `rdfs:class` and `rdfs:subClassOf`), associate properties with classes (using `rdfs:domain`), and define the range of property's values (using `rdfs:range`). The root of a class hierarchy is always `rdfs:Resource`.

Figure 3 represents the class hierarchy for the RDF Schema (Brickley & Guha, 1999). Properties are defined by `rdf:Property` and are resources themselves. RDF Schema defines four properties that are used to define other RDF Schema constructs. These are: `rdfs:subClassOf`, `rdf:type`, `rdfs:range`, and `rdfs:domain`. The “`rdf`” prefix refers to RDF namespace, and “`rdfs`” refers to the RDF Schema namespace.

RDF Schema defines types similar to object-oriented design. However, it does not define attributes in the class itself. Instead, each predicate (attribute) states which classes are its subjects.

RDF defines the structure that The Semantic Web uses to maintain the Web knowledge. RDF Schema can be used to define a new vocabulary to describe Web resources.

Figure 3 - Class Hierarchy for RDF Schema

3.1.3.3 RDF Example

The following section provides a simple RDF and RDF Schema example illustrated in RDF labeled graph and XML syntax. Consider the following sentence:

Sam is the producer of a play called "The End."

This sentence can be illustrated in RDF labeled graph as shown in Figure 4.

Figure 4 - RDF Labeled Graph Example 2

The above diagram shows that the resource “http://www.example.org/plays/TheEnd” has three attributes, namely: “Title,” “Producer,” and “Type.”

The RDF/XML syntax for the above diagram is:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf = http://www.w3.org/TR/WD-rdf-syntax#
        xmlns:a = http://example.org/play#>
    <rdf:Description about = "http://example.org/plays/TheEnd">
        <rdf:type rdf:resource=http://example.org/play#Play"/>
        <a:Producer> Sam </a:Producer>
        <a:Title> "The End" </a:Title>
    </rdf:Description>
</rdf:RDF>
```

The first line describes the XML declaration. Next, the `<rdf:RDF>` element marks the beginning of the RDF data model. The “`xmlns:rdf = http://www.w3.org/TR/WD-rdf-syntax#`” and “`xmlns:a = http://example.org/play#`” define the “rdf” and “a” namespaces respectively. Next, the `<rdf:Description>` element defines the resource being described. The `<rdf:type>` attribute defines the type of the resource as “`http://example.org/play#Play`.” Next, the “Producer” and “Title” attributes defined in “a” namespace are given the values “Sam” and “The End” respectively. Finally, the last two lines indicate the end of the resource and RDF data model description.

The corresponding RDF Schema is described as:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xml:base="http://example.org/play#" >
    <rdfs:Class rdf:ID="Play">
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema
            #Resource"/>
    </rdfs:Class>
    <rdf:Property rdf:ID="Producer">
```

```

<rdfs:domain rdf:resource="#Play"/>
<rdfs:range
  rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:ID="Title">
  <rdfs:domain rdf:resource="#Play"/>
  <rdfs:range
    rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
</rdf:RDF>

```

The above RDF Schema code (written in XML) defines a new class, "Play," that is derived (<rdfs:subClassOf>) from "Resource" super class. This class has two properties (attributes) namely: "Producer" and "Title." The domain (<rdfs:domain>) of the properties is of type "Play." The range of the properties (<rdfs:range>) is of type "string."

3.1.4 *Ontology Vocabulary*

Ontology, the fourth layer in The Semantic Web architecture, is a specification of a representational vocabulary for shared domain of discourse (Harmelen, Patel-Schneider, & Horrocks, 2001). Ontology is a formal description of terms and their relationships. It includes taxonomy and inference rules. Ontology's taxonomy uses RDF Schema to define properties and classes. It also can be used to assign properties to classes and allow subclasses to inherit these properties. The result is increasing the relations among entities on the Web. Inference in ontologies, being able to derive new data from already known data, helps computer systems manipulate data more effectively.

Various languages such as RDF Schema, Defense Advanced Research Projects Agency Agent Markup Language With Ontology Inference Layer (DAML+OIL) (Harmelen, Patel-Schneider, & Horrocks, 2001), or Web Ontology Language (OWL) (W3c, 2003) can express ontologies.

3.1.5 Logic, Proof, and Trust

Logic:

Logic is the fifth layer in The Semantic Web architecture. It is a way to use rules and make inference out of RDF statements.

Proof:

Proof is the sixth layer in The Semantic Web architecture. It puts the logical rules and statements together to draw conclusions, or in other words, proofs.

Trust:

This is where Digital Signatures (Desautels, 1999) come in. Digital signatures provide proof that a certain person wrote a document or a statement. An application chooses which signatures to trust and which not to. Since each can set its own levels of trust, the application can decide how much of what it reads to be valid.

4. Fuzzy Logic

Fuzzy logic, introduced by Zadeh 1965, is an extension of the Boolean logic to handle the truth values beyond either “completely true” and completely false.” For

example, the Fuzzy statement “Sam is old “ contains imprecision in age description.

What is the correct age to be considered as old and what to be considered as young?

Fuzzy logic can answer the above imprecise question. The range of ages to be considered old is not completely separated from those ages considered young. Figure 5 illustrates the concept of old age as a continuous curve spanning across different ages.

In accordance with the membership function as depicted in Figure 5, we can answer the question addressed. The statement on “Sam is old” might be 100% true if Sam is 75 years old, 80% true if Sam is 65 years old, 50% true if Sam is 55 years old, and 0% true if Sam is 20 years old.

Figure 5 - Membership Function for "Old" Age

4.1 Fuzzy Set versus Crisp Set

In classical set theory, the boundaries are sharply defined between members that belong to the set and those that are not. For example, the set X of old aged $\{X/X \geq 35\}$

consider only those ages greater or equal to 35 to be members of the set X, and the other ages to be non-members of the set X. The membership graph for the set X under crisp logic is illustrated in Figure 6. The degree of membership is either 1 (member) or 0 (non-member). An age of 39 is considered old.

On the other hand, Fuzzy set theory groups data with boundaries that are not sharply defined. For example, the Fuzzy set Y of old ages in the set U of all ages can be expressed as a set of ordered pair:

$$Y = \{(x, m_y(x)) / x \in U\}$$

Where $m_y(x)$ is the membership function that gives the degree of membership of X in set Y.

Figure 6 - Crisp Membership for "Old" Age

Figure 5 illustrates a possible membership function for $m_y(x)$. The degrees of membership are in the range 0 (non-membership) to 1 (full-membership). From

Figure 5, an age of 35 is considered “old” at a degree of 0.2 (20%) with respect to this membership function.

4.2 Linguistic Variable

A linguistic variable is a variable whose value is not a number but a word. For example, the linguistic variable “age” might have the linguistic value “very young,” “young,” “old,” or “very old.” Each linguistic value is associated with a Fuzzy set, each of which has a defined membership function. A membership function describes these linguistic values in terms of numerals. The linguistic variables and their membership functions allow Fuzzy logic to perform the imprecise and non-numerical reasoning performed by humans.

4.3 Membership Functions

Each Fuzzy set is associated with a membership function. There are many types of membership functions. Some of the common ones are:

1. Triangular:

Figure 7 shows a triangular shaped curve that can be described by three points, namely: $(x_1, 0)$, $(x_2, 1)$, and $(x_3, 0)$. The equation for this curve is described in (4.1):

$$F(x) = \max \left(\min \left(\frac{x - x_1}{x_2 - x_1}, \frac{x_3 - x}{x_3 - x_2} \right), 0 \right) \quad (4.1)$$

2. Trapezoidal:

Figure 8 shows a trapezoidal shaped curve that can be described by four points, namely: $(x_1, 0)$, $(x_2, 1)$, $(x_3, 1)$, and $(x_4, 0)$. The equation for this curve is described in (4.2):

$$F(x) = \max \left(\min \left(\frac{x - x_1}{x_2 - x_1}, 1, \frac{x_4 - x}{x_4 - x_3} \right), 0 \right) \quad (4.2)$$

Figure 7 - A Triangular Shaped Curve

Figure 8 - A Trapezoidal Shaped Curve

3. Gaussian

Figure 9 shows a bell shaped curve. The width of the curve is determined by σ . The center is determined by $\bar{x}$. The equation is described in (4.3):

$$F(x) = e^{-\frac{(x-\bar{x})^2}{2\sigma^2}} \quad (4.3)$$

Figure 9 - A Gaussian Shaped Curve

4.4 Fuzzy Inference System

Fuzzy systems are based on Fuzzy Set Theory and Fuzzy Logic. They can be applied to systems that support human decision making and whose information is inherently fuzzy. Example systems are: air conditioning, washing machine, traffic light system, and more.

Fuzzy systems has four major components as shown in the following architecture (Figure 10):

The following is a brief description of each component:

- **Fuzzification:**

Each linguistic variable, e.g., “age,” is associated with membership functions that are defined for the affiliated Fuzzy sets e.g., “young” and “old.” This component receives a crisp input value for each linguistic variable, and evaluates the degree of membership of this variable with respect to each affiliated Fuzzy set.

Figure 10 - Fuzzy Inference System Components

- **Fuzzy Rule Base**

Fuzzy systems use Fuzzy IF-THEN rules. A Fuzzy IF-THEN rule is of the form:

$$\text{IF } X_1 = A_1 \text{ and } X_2 = A_2 \dots \text{ and } X_n = A_n \text{ THEN } Y = B$$

Where X_i and Y are linguistic variables and A_i and B are linguistic values. The IF part of a rule is known as the antecedent, while the THEN part is known as the consequence or conclusion. The collection of Fuzzy IF-THEN rules is stored in the Fuzzy Rule Base, which is accessed by the Fuzzy Inference Engine when inputs are being processed.

- **Fuzzy Inference Engine**

This component receives all the crisp input values that have been fuzzified with respect to their respective linguistic values, accesses the Fuzzy Rule Base, and consequently derives linguistic values for the intermediate, as well as, the output linguistic variables. The two main steps in the inference process are aggregation and composition. Aggregation is the process of computing for the membership values of the IF (antecedent) part of the rules, while composition is the process of computing for the membership values of the THEN (conclusion) part of the rules. During aggregation, each condition in the IF part of a rule is assigned a degree of truth based on the degree of membership of the corresponding linguistic value. From here, either the minimum or product of the degrees of truth of the conditions is usually computed to clip the degree of truth of the IF part. This is assigned as the degree of truth of the THEN part. To determine the degrees of truth of each linguistic value of the output linguistic variable, usually, either the maximum or sum of the degrees of truth of the rules with the same linguistic values in the THEN parts is computed.

Some of the common operations that can be applied for aggregation and composition are:

- **Min-Max**

This method uses the minimum operation for the aggregation and the maximum operation for the composition. For example, consider the following rules:

Rule 1: IF x is A1 and y is B1 THEN c is C1

Rule 2: IF x is A2 and y is B2 THEN c is C2

Where x, y, and c are linguistic variables. A_i , B_i , and C_i , $1 \leq i \leq 2$, are linguistic values (with membership functions). The Min-Max operation is illustrated in Figure 11.

- Min

This operation finds the minimum degree of membership of the given fuzzy sets.

- Max

This operation finds the maximum degree of membership of the given fuzzy sets.

Figure 11 - The Min-Max Operation

- **Defuzzification**

The process of converting the fuzzified values back to crisp values is called defuzzification. There are different ways to defuzzify. Some common ones are as follows:

- **Centroid:**

This method uses weighted means to locate a centroid point for the fuzzy solution. It is computed as described in (4.4):

$$x = \frac{\sum_{j=0}^n w_j Y_A(w_j)}{\sum_{j=0}^n Y_A(w_j)} \quad (4.4)$$

Where A is a fuzzy solution, n is the number of quantization levels of the output, w_i is the i th domain value, and $Y(d)$ is the membership value for that domain.

- **Maximum Height:**

This method uses the highest point in the fuzzy region as the output solution.

An example is shown in Figure 12.

Figure 12 - The Maximum Height Method of Defuzzification

To illustrate how information is processed in a Fuzzy Inference System consider the following simple example:

4.5 A “Tipper” Fuzzy System

The “Tipper” Fuzzy system is designed to advice about the amount of tip to give in a restaurant. The decision is based on the quality of food and service. This system has two linguistic inputs: service and food. The “service” linguistic variable has three linguistic values: poor, good, and excellent. Each linguistic value is associated with a membership function as described in Figure 13. The service input is evaluated as a number between 0 and 10, where 0 indicates a poor service and 10 indicates an excellent service.

The “food” linguistic variable has two linguistic values: rancid and delicious. Each linguistic value is associated with a membership function as shown in Figure 14. The food input is evaluated as a number between 0 and 10, where 0 indicates a rancid food and 10 indicates a delicious food.

Figure 13 - Service Membership Functions

The Rule Base consists of the following three rules:

Rule 1: **IF** service is poor and food is rancid **THEN** tip is cheap

Rule 2: **IF** service is good **THEN** tip is average

Rule 3: **IF** service is excellent and food is delicious **THEN** tip is generous

The system has one output, “tip.” It is a linguistic variable with three linguistic values: cheap, average, and generous. Each linguistic value is associated with a membership function as shown in Figure 15. The tip ranges between 0 and 30.

First, the system performs the Fuzzification step on both inputs and then passes the result to the Fuzzy Inference Engine. Consider food is evaluated as 8 and service as 6. Then, the Fuzzification step returns the membership degree of each input to its corresponding Fuzzy sets as follows:

- From Figure 13:
 - Poor (service = 6) = 0

- Good (service = 6) = 0.6
- Excellent (service = 6) = 0
- From Figure 14:
 - Rancid (food = 8) = 0
 - Delicious (food = 8) = 0.4

Figure 14 - The Food Membership Functions

Then, the Fuzzy Inference Engine accesses the Fuzzy Rule Base, and derives linguistic values for the intermediate and the output linguistic variables.

Figure 16 - Min-Max Operation for the Tipper Example

This example uses the Min-Max operation for aggregation and composition as shown in Figure 16. It uses Max operation for the Defuzzification step. Figure 16 concludes that the final recommended tip is about \$18. This example illustrates how Fuzzy systems process vague information to make a human like decision.

5. Incorporating Fuzzy Logic into The Semantic Web – a New Approach to Process Perception Based Information

Usually, Web knowledge is stored in a crisp way where there is no uncertainty in the definitions or descriptions of such knowledge. For example, the statements “Zamty is a river” and “Zamty is 2600 kilometer long” clearly states that “Zamty” is of type “River” and has a length of 2600 kilometer. The existing Semantic Web is capable of providing answers to questions such as “How long is Zamty?” “What is Zamty?” and “Which river is 2600 kilometer long?” However, it cannot answer the following questions: “Is Zamty a long river?” “Which river is long?” and “Is Zamty a short river?”

In order to answer such questions, a Fuzzy system is needed to process Web information in a way that can give answers whether Zamty is long, very long, or a short river.

Section 5.1 proposes a scheme to represent Fuzzy systems in The Semantic Web using RDF. Section 5.2 shows how Fuzzy system components represented in RDF interact together. Finally, section 5.3 provides an example to illustrate the proposed scheme.

5.1 Fuzzy System Representation in The Semantic Web

The Semantic Web uses RDF to define the structure for maintaining Web knowledge. Figure 17 proposes a Fuzzy system description in RDF labeled graph. Using this description, Fuzzy systems can be represented in The Semantic Web.

A Fuzzy system is represented as an RDF resource with four properties, namely: Rule Set, Defuzzification Method, Membership Functions Repository, and Logical Operators. The representation uses RDF labeled graph as shown in Figure 17. The components in Figure 17 are:

- Rule Set:** A collection of Fuzzy “IF-THEN” rules. It is represented as an RDF Bag (rdf:Bag) of rules. Each rule has three properties, namely: “IF” that represents the antecedent part of the rule, “THEN” that represents the consequent part of the rule, and “WEIGHT” that represents the rule’s priority. See Figure 18 and Figure 19 for different rule representations in RDF. Figure 18 describes “Rule” as a resource with three properties: “IF,” “THEN,” and “WEIGHT.” Every two antecedents or consequents have a logical relation represented in “AND” or “OR” logical operators that can be defined in RDF and XML. Figure 19, describes the “IF” and “THEN” parts of a rule as an RDF Seq (rdf:Seq). Each rule’s antecedent and consequent can be represented in RDF Statement (rdf:Statement) with Subject (rdf:subject), Predicate (rdf:predicate), and Object (rdf:object). The Object is a Fuzzy set (that describes the Predicate) with membership function that is defined in XML.

- **Defuzzification Method:** It is an optional Fuzzy component that can be represented as one of the following:
 - RDF typed literal using XML Schema string data type.
 - RDF URI reference to XML document that describes the defuzzification method/function.
- **Membership Function Repository:** This component is used as a storage unit for all the membership functions used to describe the Fuzzy system. It is represented as an RDF Bag of membership functions. Each can be described in XML using the following proposed XML Schema data types:
 - **Function:** An XML Schema complex data type (shown in Figure 20) that defines the content and structure of a “Function” element. It consists of either a set of “user-defined-function” elements, or a “pre-defined-function” element. It also has four attributes, namely: “Name,” “LowerRange,” “UpperRange,” “LowerDomain,” and “Upper-Domain.”
 - **User-defined-function:** An XML Schema complex data type (shown in Figure 21) that defines the content and structure of a “user-defined-function” element. It consists of either a “Constant” element, or a set of “Operator” elements. It also has four attributes, namely: “Name,” “LowerRange,” “UpperRange,” “LowerDomain,” and “Upper-Domain.”

- **Pre-defined-function:** An XML Schema complex data type that defines the content and structure of a “pre-defined-function” element. It consists of a “Type” element that can be trapezoid (tramf), gaussian (gaussmf), or any other known membership function (Appendix C contains the schema code).
- **Operator:** An XML Schema complex data type that defines the structure and content of an “Operator” element. It can be one of the following mathematical operations:
 - Arithmetic operations (summation, multiplication, minus, power, and round)
 - Logarithmic operations (exp, and log)
 - Trigonometric operation (sin, cos, tan, sec, cosec, etc.)
 - Condition operation (if)

These operations can be used to describe any function (Appendix B contains the schema code).

- **Logical Operators:** This component describes all the operators used to evaluate the Fuzzy rules. There are three common operators: “AND,” “OR,” and “NOT,” that can be described using RDF typed literal or RDF URI reference.

Figure 17 - Fuzzy System Description in RDF Labeled Graph

Figure 18 - RDF for Fuzzy IF-THEN Rule (1)

Figure 19 - RDF for Fuzzy IF-THEN Rule (2)

```
<xs:complexType name="Function">
  <xs:choice>
    <xs:element name="user_defined_function" type="user_defined_function" maxOccurs="unbounded"/>
    <xs:element name="pre_defined_function" type="pre_defined_function"/>
  </xs:choice>
  <xs:attribute name="Name" type="xs:string" use="optional"/>
  <xs:attribute name="LowerRange" type="xs:double" use="optional"/>
  <xs:attribute name="UpperRange" type="xs:double" use="optional"/>
  <xs:attribute name="LowerDomain" type="xs:double" use="optional"/>
  <xs:attribute name="UpperDomain" type="xs:double" use="optional"/>
</xs:complexType>
```

Figure 20 – “FUNCTION” XML Schema

A key advantage of the above RDF description is that it can be used to exchange and integrate Fuzzy systems information with other Semantic Web applications that use Fuzzy logic in their solutions. This enables distributed Fuzzy based applications in The Semantic Web.

```

<xs:complexType name="user_defined_function">
  <xs:choice>
    <xs:element name="Constant">
      <xs:complexType>
        <xs:choice>
          <xs:element name="integer_number" type="xs:integer"/>
          <xs:element name="float_number" type="xs:float"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="Operator" type="Operator" maxOccurs="unbounded"/>
  </xs:choice>
  <xs:attribute name="Name" type="xs:string" use="optional"/>
  <xs:attribute name="LowerRange" type="xs:double" use="optional"/>
  <xs:attribute name="UpperRange" type="xs:double" use="optional"/>
  <xs:attribute name="LowerDomain" type="xs:double" use="optional"/>
  <xs:attribute name="UpperDomain" type="xs:double" use="optional"/>
</xs:complexType>

```

Figure 21 – “User-defined-function” XML Schema

5.2 Fuzzy System Components in Semantic Web Applications

This section proposes architecture for integrating Fuzzy system in The Semantic Web. The proposed architecture allows for processing perception-based information in The Semantic Web. Specifically, it extends the current Semantic Web to accept Fuzzy input represented in RDF and XML, and use this input along with existing crisp input to process perception based queries. Figure 22 shows how RDF integrates and interacts with Fuzzy inference system to process perception-based user input, and produce an output. The Fuzzy inference system, as well as, the object data is described using RDF.

The Fuzzy Inference System first finds the set of all objects that match the crisp user input. Then, it finds the most relevant objects to the perception based user input. A more detailed description of the interactions between RDF and the Fuzzy inference system is shown in Figure 23.

Figure 23 shows five steps (components) of the Fuzzy Inference System. These steps are described below:

- **Step 1:** The system narrows the search space based on crisp user input and crisp object data. The result is a reduced set of objects O to search on.

$$O = [O_1, \dots, O_n] \quad (5.1)$$

Each object O_i is described by a set of properties P_i :

$$P_i = [P_{i1}, \dots, P_{in}] \quad (5.2)$$

Where P_{ij} is the value of property P_j in object O_i .

The object data consists of:

- Class hierarchy in RDF Schema. This Schema defines the class hierarchy in the system and the attributes of each class.
 - Crisp data in RDF.
- **Step 2:** The system performs Fuzzification process using the set of objects O defined in (5.1), and the Fuzzy sets FS :

$$FS = \{FS_{P1}, \dots, FS_{Pn}\} \quad (5.3)$$

$$FS_{P_i} = \{FS_{P_{i1}}, \dots, FS_{P_{im}}\} \quad (5.4)$$

$FS_{P_{ij}}$, represents a Fuzzy set j that describes the property P_i . Each Fuzzy set $FS_{P_{ij}}$ has a membership function MF_{ij} associated with it. The Property hierarchy RDF Schema defines the structure of the Fuzzy sets. This Schema extends the definition of object properties P_i , as defined in (5.2), to linguistic variables with Fuzzy sets. The “Function” XML Schema (refer to Figure 20) defines the structure of the membership function in the Fuzzy set.

Figure 22 - Fuzzy System Components in The Semantic Web – A High Level Description

The Fuzzification process maps the crisp input (object properties P_i as described in (5.2)) to the appropriate Fuzzy sets, and calculates the degree of membership for each corresponding set. For example, it maps the value “2600” from “Zamty is 2600 kilometer” example to “long,” “short,” and “medium” Fuzzy sets, and calculates the degree of membership for “length 2600” as (0.8) long, (0.6) medium, and (0.4) short.

For each object O_i and for each property P_{ij} , the Fuzzification process returns a set of values (between 0 and 1) that correspond to the membership degree of the

property value to each Fuzzy set that belongs to this property. This membership degree value is calculated using the membership function of the Fuzzy set along with the crisp value of the property provided by the object. These values are passed to the Fuzzy Inference process described in Step 3, and to the Output process described in Step 5.

- **Step 3:** The Fuzzy Inference process receives the “Fuzzification Process” output, the Fuzzy rule set, and the logical operators functions. For each object O_i , it evaluates all the rules using the object’s membership degree values calculated in Step 2, and the functions that define the logical operators used in the rule set. Each rule evaluated returns the truth-value of its consequent part (Fuzzy set value). Then, all the rules truth-values are passed to the Defuzzification process described in Step 4.

The Fuzzy rule set consists of IF-THEN rules. These rules are used to make inferences about new properties. The following is an RDF example that describes the rule “IF (river is long) THEN (river is a medium stream)”:

1. (rl:rule, ex:weight, 10)
2. (_:antecedent, rdf:type, rdf:statement)
3. (_:antecedent, rdf:subject, nt:river)
4. (_:antecedent, rdf:predicate, at:length)
5. (_:antecedent, rdf:object, at:long)
6. (rl:rule, rl:if, _:antecedent)
7. (_:c1, rdf:type, rdf:statement)
8. (_:c1, rdf:subject, nt:stream)
9. (_:c1, rdf:predicate, at:length)
10. (_:c1, rdf:object, at:medium)
11. (_:consequent, rdf:type, rdf:statement)
12. (_:consequent, rdf:subject, nt:river)
13. (_:consequent, rdf:predicate, exm:is)
14. (_:consequent, rdf:object, _:c1)
15. (rl:rule, rl:then, _:consequent)

Figure 23 – Fuzzy System Components in Semantic Web – Detailed Interactions

The above rule has a weight (line 1). It has one antecedent (line 6) and one consequent (line 15). The “_:antecedent” is a statement (line 2) that has a subject (river), a predicate (length), and an object (long) (line 3, 4, and 5 respectively). This statement is used to describe the resource “rl:rule” (line 6). The consequent part is

defined in line 15, and the “_:consequent” statement is used to describe it. Figure 18 and Figure 19 suggest two RDF representations for a “Rule” resource. A set of rules can be represented in RDF bag (rdf:Bag). The functions that define the logical operators used in the rule set are defined in XML and RDF.

- **Step 4:** The Defuzzification process receives the “Inference Engine” output (for each object O_i) and the defuzzification method function (i.e., Max, Min, and Centroid), applies the defuzzification method on the “Inference Engine” output, and maps the resulted Fuzzy sets values (rules truth value) to crisp output values (new properties values). Then, it passes the values to the Output in Step 5. The defuzzification method is a function that can be defined in XML and RDF.
- **Step 5:** The perception-based input from users consists of a list of Fuzzy sets:

$$U = [U_1, \dots, U_n], \text{ where } U_i \in \text{FS} \quad (5.5)$$

Where, FS is as defined in (5.3). Let

$$MF = [MF_1, \dots, MF_n] \quad (5.6)$$

MF represents a set of membership functions that corresponds to U in (5.5).

For each object O_i with properties vector P_i , as described in (5.2), define a vector:

$$V_i = [V_{i1}, \dots, V_{in}], \text{ with } V_{ik} = MF_k(P_{ik}) \in [0 \ 1] \quad (5.7)$$

The value V_{ik} is the membership degree of P_{ik} in U_k .

The vectors V_i for all objects O are used to sort the objects according to their relevancy to the user’s query. To do so, calculate the norm of V_i as:

$$|V_i| = \sum_{k=1}^n (V_{ik})^2 \quad (5.8)$$

Then, sort the objects according to $|V_i|$. The Object with maximum $|V_i|$ value represents the best result.

The Output process receives the “Fuzzification” and “Defuzzification” outputs, builds a vector V_i for each object O_i , and calculates $|V_i|$ value as described in (5.8). Then, it sorts the set of objects O by $|V_i|$ in decreasing order. The sorted output is then presented to the user.

The following section illustrates the above methodology in a real Web application.

5.3 Implementing the Proposed Architecture- an Example

The following section uses the proposed architecture in section 5.2 to implement a School Search application. Existing Web users who search for schools that fit their needs end up with a long list of schools ordered by different categories. Some of these categories are: Academic Performance Index (API), Teacher’s Experience (TE), Percentage of Students who are African American (AA), Percentage of Students who are Asian American or Pacific Islander (APA), Average Class Size (ACS), Students per Computer (SPC), Students per Teacher (SPT), Percent of Students who have Reduced Price Lunch (RPL), and more.

The existing Web and The Semantic Web do not allow users to specify categories in natural languages, such as High API, High APA, and Low ACS. A Fuzzy system allows users to provide input in a natural language, and can process such input to provide relevant and appropriate results. The proposed soft Semantic system for this example

will recommend a list of schools ranked in order of user preference. The preference inputs are API, TE, AA, APA, etc.

Figure 24 illustrates the architecture of the “School Search” Fuzzy system in The Semantic Web.

Figure 24 – School Search Fuzzy System Architecture

This system uses the proposed architecture in section 5.2, and consists of the following steps:

Step 1: The School Search system narrows the search space based on crisp user input (County, School Type, and School Grade) and available data about schools (see Figure 25 for user input example). The result is a reduced set of schools to search on. The data about schools is represented in RDF and follows the Schema illustrated in Figure 26 (which describes the School class hierarchy). Figure 27 shows a partial RDF Schema of the School class hierarchy and its properties, such as SPC, API, AA, and APA. Figure 28 shows sample code of the school's data.

The screenshot shows a web browser window with a search interface. At the top, there is a menu bar with 'File', 'Edit', 'View', 'Favorites', 'Tools', and 'Help'. Below the menu bar is a search bar with a magnifying glass icon and a 'Search' button. The main content area contains several filters and criteria:

- County:** A dropdown menu with 'Alameda' selected.
- School Type:** A dropdown menu with 'All' selected.
- School Grade:** Three radio buttons: 'Elementary School' (selected), 'Middle School', and 'High School'.
- Academic Performance Index (API):** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Teachers experience:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Elementary teachers credentials:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Students per teacher:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Average class size:** Three radio buttons: 'Big' (selected), 'Medium', and 'Small'.
- Student per computer:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of students not yet proficient in English:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of students who are African American:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of student who are Asian or Pacific American:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of student who are Latino or Hispanic:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of student who are White, European-American, Native American or Other:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of students with low family income:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of parents with college degree:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.
- Percent of student who will attend college:** Three radio buttons: 'High' (selected), 'Medium', and 'Low'.

At the bottom of the form, there are two buttons: 'RESET' and 'SUBMIT'.

Figure 25 - School User Interface

Figure 26 - School Class Hierarchy

```

<rdfs:Class rdf:ID="SCHOOL">
  </rdfs:Class>
  <rdfs:Class rdf:ID="PUBLIC">
    <rdfs:subClassOf rdf:resource="#SCHOOL"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="PRIVATE">
    <rdfs:subClassOf rdf:resource="#SCHOOL"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="RELIGIOUS">
    <rdfs:subClassOf rdf:resource="#PRIVATE"/>
  </rdfs:Class>
  ...
<rdf:Property rdf:ID="SPC">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
  <rdfs:domain rdf:resource="#SCHOOL"/>
</rdf:Property>

<rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Datatype"/>
</rdf:Description>
  ...

```

Figure 27 – RDF/XML Sample Code for School Hierarchy

- **Step 2 (Fuzzification):** The system performs the Fuzzification process using the school's data from Step 1 and the Fuzzy sets that describe the School properties. The Property class hierarchy defines the structure of the Fuzzy sets and is illustrated in Figure 29. Figure 30 shows a sample code for the Property class hierarchy in RDF Schema. Each school property has three Fuzzy sets that describe the degree of the property (HIGH, MED, and LOW). For example, the API property has three Fuzzy sets, namely: "API_HIGH," "API_MED," and API_LOW." Each Fuzzy set has a membership function that can be defined in XML. Table 1 lists all the membership functions used in this system. For example, the Fuzzy set "API_HIGH" has a "trapezoid" membership function with four parameters (0,980,1000,1000). A trapezoid membership function with four parameters (x1, x2, x3, x4) is a trapezoidal shaped function that is described by four points, namely: (x1, 0), (x2, 1), (x3, 1), and (x4, 0). Figure 30 shows the XML code for the "API_HIGH" membership function. It is a "Function" data type, and it is represented as "pre_defined_function" of type "trapmf" (trapezoid) with four parameters.

For each School and for each property value, this process calculates the degree of membership of the property value to all the Fuzzy sets that describe the property.

For example, it takes the "API" property value 952, maps it to API_HIGH (High API), API_MED (Medium API), and API_LOW (Low API) Fuzzy sets, and then calculates the degree of membership of the property value 952 using the membership

functions described in Table 1. Then, all these values are passed to the Fuzzy Inference process described in Step 3, and to the Output process described in Step 5.

```

<dt:PRIVATE rdf:ID="Mission San Jose Elementary School">
  <dt:COUNTRY> Alameda </dt:COUNTRY>
  <dt:LEVEL> Elementary </dt:LEVEL>
  <dt:ADDRESS> 43545 Bryant St., Fremont, CA, 94545</dt:ADDRESS>
  <dt:PHONE>510-656-1200 </dt:PHONE>
  <dt:API> 952</dt:API>
  <dt:LEP> 11 </dt:LEP>
  <dt:AA> 1 </dt:AA>
  <dt:APA> 69 </dt:APA>
  <dt:LH> 2 </dt:LH>
  <dt:WO> 24 </dt:WO>
  <dt:RPL> 7 </dt:RPL>
  <dt:TE> 14</dt:TE>
  <dt:ETC> 96 </dt:ETC>
  <dt:SPT> 23 </dt:SPT>
  <dt:ACS> 20 </dt:ACS>
  <dt:SPC> 6 </dt:SPC>
</dt:PRIVATE>

```

Figure 28 – RDF/XML Data Sample

Figure 29 - Property Hierarchy for Schools

```

<rdfs:Class rdf:ID="SCHOOL_PROPERTY">
  </rdfs:Class>

  <rdfs:Class rdf:ID="API">
    <rdfs:subClassOf rdf:resource="#SCHOOL_PROPERTY"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="LEP">
    <rdfs:subClassOf rdf:resource="#SCHOOL_PROPERTY"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="AA">
    <rdfs:subClassOf rdf:resource="#SCHOOL_PROPERTY"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="APA">
    <rdfs:subClassOf rdf:resource="#SCHOOL_PROPERTY"/>
  </rdfs:Class>
  ...
<rdf:Property rdf:ID="NAME">
  <rdfs:comment> Fuzzy Set Name</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#SCHOOL_PROPERTY"/>
</rdf:Property>

<rdf:Property rdf:ID="MEMBERSHIP_FUNCTION">
  <rdfs:comment> The range is a URI for the membership function location </rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#SCHOOL_PROPERTY"/>
</rdf:Property>
...

```

Figure 30 - RDF Schema Sample Code for School Properties

- **Step 3 (Fuzzy Inference):** The Fuzzy Inference process receives the “Fuzzification Process” output, the Fuzzy rule set, and the set of logical operators and their definition. The Fuzzy rule set consists of IF-THEN rules that determine new school attributes from existing ones.

For example, “IF (API is high) and (RPL is low) THEN (percentage of parents who have college degree (PCD) is high) “ or “ IF (API is low) THEN (percentage of students who attend college (SAC) is low).” Table 2 contains a

description of the rule set. A rule description follows the Schema defined in Figure 18, while the rule set is represented in RDF Bag. This example defines the logical

```
<Function xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\XMLSCHEMA\memberFunction.xsd" Name="API_HIGH" LowerRange="0"
UpperRange="1" LowerDomain="0" UpperDomain="1000">
  <pre_defined_function>
    <Type>
      <trapmf NumOfParam="4" Name="trapmf">
        <parameter>
          <number>0</number>
        </parameter>
        <parameter>
          <number>980</number>
        </parameter>
        <parameter>
          <number>1000</number>
        </parameter>
        <parameter>
          <number>1000</number>
        </parameter>
      </trapmf>
    </Type>
  </pre_defined_function>
</Function>
```

Figure 31 - "API_HIGH" Function in XML

operator “AND” to be the “Minimum” function. For each school in the search space, this process evaluates all the rules to determine their truth-value. For example, if we apply rule number 1 (Table 2) for Mission Elementary School (Figure 28) we conclude that the truth-value for “SAC is HIGH” is $\text{Min}(\text{API_HIGH}(952) = 0.971, \text{RPL_LOW}(7) = 0.875)$ equal to 0.875. See Table 3 for the evaluation of all the rules that apply to Mission Elementary School. These values are then passed to the Defuzzification process in Step 4.

Table 1: Membership Functions for School System

School Property	Fuzzy Sets					
	Name	MF	Name	MF	Name	MF
Academic Performance Index	HIGH	Trap (0,980,1000,1000)	MED	Trap (200,820,870,960)	LOW	Trap (0,0,200,800)
Teacher's Experience	HIGH	Trap (0,20,45,45)	MED	Trap (0,4,6,9)	LOW	Trap (0,0,1,6)
Elementary Teachers Cardinality	HIGH	Trap (29,95,100,100)	MED	Trap (20,80,85,100)	LOW	Trap (0,0,25,75)
Student Per Teacher	HIGH	Trap (0,5,26,40)	MED	Trap (0,3,20,23)	LOW	Trap (0,0,7,22)
Average Class Size	BIG	Trap (10,23,40,40)	MED	Trap (0,14,18,23)	SMALL	Trap (0,0,10,18)
Student Per Computer	HIGH	Trap (2,20,90,90)	MED	Trap (0,4,8,20)	LOW	Trap (0,0,1,8)
Percent Of Students Not Yet Proficient In English	HIGH	Trap (5,40,90,90)	MED	Trap (0,10,15,30)	LOW	Trap (0,0,3,15)
Percent Of Student who are African American	HIGH	Trap (5,80,100,100)	MED	Trap (0,45,55,75)	LOW	Trap (0,0,3,45)
Percent Of Student Who are Asian or Pacific American	HIGH	Trap (5,80,100,100)	MED	Trap (0,45,55,75)	LOW	Trap (0,0,3,45)
Percent Of Students Who are Latino or Hispanic	HIGH	Trap(5,80,100,100)	MED	Trap (0,45,55,75)	LOW	Trap (0,0,3,45)
Percent Of Student Who are White or Others	HIGH	Trap (5,80,100,100)	MED	Trap (0,45,55,75)	LOW	Trap (0,0,3,45)
Percent Of Students With Low Family Income	HIGH	Trap (10,60,100,100)	MED	Trap (2,15,30,50)	LOW	Trap (0,0,3,35)
Percent Of Students Who Will Attend College	HIGH	Trap (20,85,100,100)	MED	Trap (5,45,55,80)	LOW	Trap (0,0,2,40)
Percent Of Parents With College Degree	HIGH	Trap (20,85,100,100)	MED	Trap (5,45,55,80)	LOW	Trap (0,0,2,40)

A membership function described as Trap (x_1 , x_2 , x_3 , x_4), is a trapezoidal shaped function (Figure 8), that is defined by four points: namely, (x_1 , 0), (x_2 , 1), (x_3 , 1), and (x_4 , 0).

Table 2: IF-THEN Rule Set for the School System

Rule Number	Fuzzy IF-THEN Rule
1	IF (API is HIGH) AND (APA is HIGH) THEN (SAC is HIGH)
2	IF (API is LOW) THEN (SAC is LOW)
3	IF (API is HIGH) AND (RPL is HIGH) THEN (SAC is MED)
4	IF (API is HIGH) AND (RPL is LOW) THEN (PCD is HIGH)
5	IF (RPL is HIGH) THEN (PCD is LOW)
6	IF (API is MED) AND (RPL is LOW) THEN (PCD is MED)

- **Step 4 (Defuzzification):** This process receives the “Fuzzy Inference” output (for each school), applies the defuzzification method (chosen as Maximum), and maps the resulted

Fuzzy set value to crisp value that represents the new property value. Then, it passes the values for each school to the Output process described in Step 5. For example, the Defuzzification process receives Table 3 from Step 3, applies the maximum function on rule 1, 2, and 3 that correspond to the SAC property, and concludes that SAC is 75.4% by using the SAC_HIGH membership function described in Table 1.

Table 3: Rule Evaluations for Mission School

Rule Number	Truth Value – Mission San Jose Elementary School
1	0.853
2	----
3	0
4	0.875
5	----
6	0.0888

- Step 5:** The Output process receives the “Fuzzification” and “Defuzzification” outputs, and for each school i in the search space it builds a vector V_i as described in (5.7). Then, it calculates $|V_i|$ as defined in (5.8). Then, it sorts the schools according to their $|V_i|$ value. For example, if $U = [\text{API_HIGH}, \text{TE_HIGH}, \text{ETC_HIGH}, \text{SPT_LOW}, \text{ACS_MED}, \text{SPC_LOW}, \text{SPE_MED}, \text{AA_LOW}, \text{APA_HIGH}, \text{LH_LOW}, \text{WO_MED}, \text{RPL_LOW}, \text{SAC_MED}, \text{PCD_MED}]$, then V for Mission School (described in Figure 28) is $[0.971, 0.7, 1, 0, 0.6, 0.2857, 1, 1, 0.853, 1, 0.533, 0.875, 0.184, 0.125]$, $|V|$ is 7.7, and the relevancy index, defined as $|V|$ divided by the number of properties, is $7.7/14$ or 55%. Figure 32 shows a possible output of this process. In this example, Mission School has

the highest $|V|$ among schools in the search space, and it is listed as the first school to match the user's query. For each school, Figure 32 lists the school's properties, their values, the perception-based user selection, the vector V as "Property Relevancy," and the overall relevancy index.

This completes the description of the School Search example using the architecture described in section 5.2. The example shows how the proposed architecture can be used to process perception-based information. Provided that the membership functions can be designed properly, this system provides a powerful tool to enable users to search efficiently and filter out redundant information usually present with current search tools. However, it should be noted that the design of the membership functions greatly affects the accuracy of the results. As such, a great effort is needed in designing these functions.

Mission San Jose Elementary School				Fred W. Weibel Elementary School			
45545 Bryant St. Fremont, CA 94545 (510-466-1300)				45555 Grimmer Blvd. Fremont, CA 94555 (510-666-6078)			
Overall Relevancy: 56%				Overall Relevancy: 64%			
School Property	Property Value	User Preference	Property Relevancy	School Property	Property Value	User Preference	Property Relevancy
Academic Performance Index	952	HIGH	0.971	Academic Performance Index	951	HIGH	0.9704
Teacher's Experience	14	HIGH	0.7	Teacher's Experience	8	HIGH	0.4
Elementary teachers' cardinality	96%	HIGH	1	Elementary teachers' cardinality	100%	HIGH	1
Student Per Teacher	23	LOW	0	Student Per Teacher	21	LOW	0.0666
Average Class Size	20	MED	0.6	Average Class Size	22	MED	0.2
Student Per Computer	6	LOW	0.2857	Student Per Computer	5	LOW	0.4285
Percent of students not yet Proficient in English	11%	MED	1	Percent of students not yet Proficient in English	11%	MED	1
Percent of Students Who are African American	1%	LOW	1	Percent of Students Who are African American	1%	LOW	1
Percent of Students Who are Asian or Pacific American	69%	HIGH	0.853	Percent of Students Who are Asian or Pacific American	82%	HIGH	1
Percent of Students Who are Latino or Hispanic	2%	LOW	1	Percent of Students Who are Latino or Hispanic	2%	LOW	1
Percent of Students Who are White or Others	24%	MED	0.533	Percent of Students Who are White or Others	15%	MED	0.333
Percent of Students With Low Family Income	7%	LOW	0.875	Percent of Students With Low Family Income	1%	LOW	1
Percent of Students Who Will Attend College	75.4%	MED	0.184	Percent of Students Who Will Attend College	83%	MED	0
Percent of Parents With College Degree	76.875	MED	0.125	Percent of Parents With College Degree	83%	MED	0

Figure 32 - School System Possible Output

6. Conclusion

This report shows how perception-based information can be processed in Semantic Web applications. It proposes a scheme to integrate Fuzzy logic in The Semantic Web.

Semantic Web application based on Fuzzy logic greatly enhances user experience. It allows the user to enter queries using natural languages. This report introduces a real life example system that allows users to search for a school based on perception-based properties. The example application demonstrates the solution proposed in this report as well as allows for experiencing with such system.

References

- Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). The Semantic Web. *Scientific American*, 284(5), 34-43.
- Brickely, D., & Guha, R.V. (1999, March). *Resource Description Framework (RDF) Schema Specifications, W3C proposed Recommendation*. Retrieved October 3, 2002, from <http://www.w3.org/TR/1999/PR-rdf-schema-19990303/>
- Brickely, D., & Guha, R.V. (2003, January). *RDF Vocabulary Description language 1.0: RDF Schema, W3C Working Draft*. Retrieved February 2, 2003, from <http://www.w3.org/TR/2003/WD-rdf-schema-20030123/>
- Choi, D. (2002, April). Enhancing the power of Web search engines by means of fuzzy query. *Decision Support Systems*, 35(1) 31-44.
- DesAutels, P.F. (1999, June). *W3C Digital Signature Initiative Overview*. Retrieved January 30, 2003, from <http://www.w3.org/DSig/>
- Dumbill, E. (2001, March). *Building the Semantic Web*. Retrieved October 16, 2002, from <http://www.xml.com/pub/a/2001/03/07/buildingsw.html>
- Fontana, F.A. (2001). Evaluation of SGML-based Information through Fuzzy Techniques. *Information Processing and Management*, 37(1), 75-90.
- Fontana, F.A. (2002). A similarity-based resolution rule. *International Journal of Intelligent Systems*, 17 (9), 853-872.
- Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2), 199-220.
- Harmelen, F., Patel-Schneider, P.F., & Horrocks, I. (2001, March). *Reference Description of DAML+OIL (March 2001) Ontology Markup Language*. Retrieved January, 2003, from <http://www.daml.org/2001/03/reference.html>
- Hunter, J., & Lagoze, C. (2001, May). *Combining RDF and XML Schemas to Enhance Interoperability Between Metadata Application Profiles*. Retrieved November 3, 2002, from <http://www10.org/cdrom/papers/572/>
- Janikow, C. Z. (1998, February). Fuzzy Decision Trees: Issues and Methods. *IEEE Transactions On Systems, Man, and Cybernetics – Part B: Cybernetics*, 28(1), 1-14.

- Lassila, O., & Swick, R.R. (1999, February). *Resource Description Framework (RDF): Model and Syntax Specification, W3C Recommendation*. Retrieved October 3, 2002, from <http://www.w3.org/TR/REC-rdf-syntax/>
- Sarkar, M. (2002). Rough – Fuzzy function in classification. *Fuzzy Sets and Systems*, 132, 353 – 369.
- W3C. (1999). *Namespaces in XML*. Retrieved October 16, 2002, from <http://www.w3.org/TR/REC-xml-names>
- W3C.(1999). *XML Schema tutorial*. Retrieved September 20, 2002, from <http://www.w3schools.com/schema/default.asp>
- W3C.(1999). *XML tutorial*. Retrieved September 20, 2002, from <http://www.w3schools.com/xml/default.asp>
- W3C. (1999). *XSL Transformation (XSLT) Version 1.0 W3C Recommendation*. Retrieved September 20, 2002, from <http://www.w3.org/TR/xslt.html>
- W3C. (2003). *OWL Web Ontology Language Reference, W3C Working Draft*. Retrieved April 12, 2003, from <http://www.w3.org/TR/owl-ref/>
- Zadeh, L.A. (1989,March). Knowledge Representation in Fuzzy Logic. *IEEE Transactions On Knowledge and Data Engineering*, 1(1), 89-100.
- Zadeh, L.A. (2001, January). Toward a perception-based theory of probabilistic reasoning with imprecise probabilities. *Journal of Statistical Planning and Inference*, 105, 233-264.

Appendices

Appendix A – Generalized Constrained Language (GCL) and The Semantic Web

GCL Overview

Many applications use probability theory, in the absence of precise and complete knowledge, to automate the process of decision-making and reasoning. Traditional probability theory has the capabilities only to operate on measurement-based information, but not on perception-based information. For example: “Usually Robert returns from work at about 6 pm. What is the probability that Robert is home at 6:30 pm? ” Cannot be answered by probability theory. Perception-based probability theory enhances the ability of probability theory to deal with a mix of measurement-based and perception-based information. It adds new capabilities to represent the meaning of propositions expressed in a natural language.

Precisiated Natural Language (PNL) is a subset of the Natural Language (NL), which consists of propositions drawn from natural language that can be precisiated through translation into Generalized Constraint Language (GCL). An example of proposition in PNL is: “It is likely to rain tomorrow.” If P is a proposition in natural language, then precisiation of P results in a representation of the meaning of P in the form of what is referred to as a GCL (Zadeh, 2001). A GCL is expressed as $X \text{ isr } R$, where X is the constrained variable, R is the constraining relation and r is the indexing variable whose value defines the way in which R constrains X.

The types of constraints and the associated values of r are as the following:

- Equality constraint ($r = "="$); $X = R$
- Inequality constraint ($r = "<="$); $X \leq R$
- Subsethood constraint ($r = "\subset"$); $X \subset R$
- Possibilistic constraint ($r = \text{blank}$); R is the possibility distribution of X
- Veristic constraint ($r = v$); R is the verity distribution of X
- Probabilistic constraint ($r = p$); R is the probability distribution of X
- Usuality constraint ($r = u$); usually X is R
- Random set constraint ($r = rs$); R is the Fuzzy-set-valued probability distribution of X
- Fuzzy graph constraint ($r = fg$); X is a function and R is its Fuzzy graph
- Pawlak set constraint ($r = ps$); X isps ($\overline{X}, \underline{X}$) - means that X is a set and $\overline{X}$ and $\underline{X}$ are the lower and upper approximation to X

The set of all the composite generalized constraints and their associated rules of generation and interpretation (combination, modification, and qualification) constitute the GCL elements. GCL serves to represent the meaning of perception-based information in perception-based probability theory.

GCL and The Semantic Web

The following section shows how perception-based information described in Generalized Constrained Language (GCL) can be represented in RDF format. This allows

utilizing existing framework, namely GCL, for inference-based systems. This work has been developed alongside the main subject of this thesis.

The Semantic Web uses RDF format to describe Web resources and their relationships to each other. Existing RDF based languages deal with hard semantics that are suitable for describing crisp statements. For example: “Mary wrote a book” and “Carol is 25.” The property value is crisp and well defined (book, and 25). RDF based languages do not have the ability to represent soft semantics (which deal with perception description inherent in natural languages). For example: “Mary is likely to write a book” or “Carol is young.”

The statements: “Mary is likely to write a book” and “Carol is young” can be represented technically as: “Likelihood (Mary to write a book) is likely” and “Age (Carol) is young,” respectively. The above descriptions can be viewed as a General Constrained Language (GCL) description. In GCL, a proposition, P , can be described in the form of $X \text{ isr } R$, where X is the constrained variable, R is the constraining relation and r is indexing variable whose value defines the way in which R constrains X .

In this work, the perception-based semantic representation capability with GCL is enabled on data represented in RDF. In particular, it transforms semantic description of GCL statements, i.e., $X \text{ isr } R$, to RDF format (triple form). This concept is illustrated with the following examples:

- **Equality constraint “ $r =$ ”; $X = R$**

Consider the statement: “Carol is 25.”

Step 1: Represent the semantic statement in technical form:

Age (Carol) is 25.

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ is } R$, where X is Age (Carol) and R is 25.

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triple components for the GCL statement in 2 can be as follows:

Subject: *Carol*

Predicate: *Age*

Object: 25

See Figure A.1 for RDF graph.

Figure A.1: RDF Graph Example A.1

- **Inequality constraint “ $r: \leq$ ”; $X \leq R$**

Consider the semantic statement: “Carol is less than 30.”

Step 1: Represent the semantic statement in technical form:

Age-Range (Carol) is [1 29].

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ is } R$, where X is Age-Range (Carol) and R is [1 29].

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF statements for the GCL statement in 2 can be as follows:

RDF_1:

Subject: *Carol*

Predicate: *Age-Range*

Object: *_:ageRange* (Blank Node)

RDF_2:

Subject: *_ageRange*

Predicate: *upper-limit*

Object: 29

RDF_3:
 Subject: *_ageRange*
 Predicate: *lower-limit*
 Object: *1*

See Figure A.2 for RDF graph.

Figure A.2: RDF Graph Example A.2

- **Subsethood constraint $X: X \subset R$**

Consider the semantic statement: “Carrots contains vitamins for eyesight.”

Step 1: Represent the semantic statement in technical form:

Subset (Vitamins-In-Carrots) included in Set (All-Vitamins-good-for-eyesight)

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ is } R$, where X is Subset (Vitamins-In-Carrots-Good-For-Eyesight) and R (All-Vitamins-Good-For-Eyesight).

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF statements’ components for the GCL statement in 2 can be as follows:

Subject: *Vitamins-In-Carrots-Good-For-Eyesight (VICGFE)*

Predicate: *Subset*

Object: *All-Vitamins-Good-For-Eyesight (AVGFE)*

“Vitamins-In-Carrots-Good-For-Eyesight” and “All-Vitamins-Good-For-Eyesight” are another RDF statements that describe `rdf:Bag` .

See Figure A.3 for RDF graph.

Figure A.3: RDF Graph Example A.3

- **Possibilistic constraint “r: blank”; X is R**

Consider the semantic statement: “Most Swedes are tall.”

Step 1: Represent the semantic statement in technical form:

Count (tall Swedes) is most.

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ is } r \text{ } R$, where X is Count (tall Swedes / Swedes) and R is most.

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triple components for the GCL statement in 2 can be as follows:

Subject: *Swedes*

Predicate: *Count-of-tall-Swedes*

Object: *most*

See Figure A.4 for RDF graph.

Figure A.4: RDF Graph Example A.4

- **Veristic constraint “r: v”; X isv R**

Consider the semantic statement: “Robert is half German, $\frac{1}{4}$ French, and $\frac{1}{4}$ Italian.”

Step 1: Represent the semantic statement in technical form:

Ethnicity (Robert) is (1/2 German, $\frac{1}{4}$ French, $\frac{1}{4}$ Italian).

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ isv } R$, where X is Ethnicity (Robert) and R is (0.5/German + 0.25/French + 0.25/Italian).

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triples components for the GCL statement in 2 can be as follows:

RDF_1:

Subject: *Robert*

Predicate: *Ethnicity*

Object: *_:Ethnicity_Bag*

RDF_2:

Subject: *_:Ethnicity_Bag*

Predicate: *rdf:type*

Object: *rdf:Bag*

RDF_3:

Subject: *_:Ethnicity_Bag*

Predicate: *rdf_1*

Object: *0.5/German*

RDF_4:

Subject: *_:Ethnicity_Bag*

Predicate: *rdf_2*

Object: *0.25/French*

RDF_5:

Subject: *_:Ethnicity_Bag*

Predicate: *rdf_3*

Object: *0.25/Italian*

See Figure A.5 for RDF graph.

Figure A.5: RDF Graph Example A.5

- **Probabilistic constraint “ $r: p$ ”; X is p R**

Consider the statement: “ X is a normally distributed random variable with mean m and variance $g2$.”

Step 1: Represent the statement in technical form:

Probability-distribution (X) is Normal (mean= m , variance= $g2$).

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

X is r R , where X is probability-distribution (X) and R is $N(m, g2)$.

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triples components for the GCL statement in 2 can be as follows:

RDF 1:

Subject: X

Predicate: *Probability-distribution*

Object: *_:normal* (Blank Node)

RDF 2:

Subject: *_:normal*

Predicate: *mean*

Object: m

RDF 3:

Subject: *_:normal*

Predicate: *variance*

Object: $g2$

See Figure A.6 for RDF graph.

Figure A.6: RDF Graph Example A.6

- **Probability-value constraint “r: pv”; X ispv R**

Consider the semantic statement: “It is likely Carol is young.”

Step 1: Represent the semantic statement in technical form:

Likelihood (Carol is young) is likely.

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ is } R$, where X is likelihood (Carol is young) and R is likely.

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triple components for the GCL statement in 2 can be as follows:

Subject: *Age (Carol) is young*

Predicate: *Likelihood*

Object: *likely*

The Subject is another RDF statement.

See Figure A.7 for RDF graph.

Figure A.7: RDF Graph Example A.7

- **Random set constraint “r: rs”; X isrs R**

Consider the semantic statement: “young is a Fuzzy set with probabilities (1/5, 0.9/25, 0.85/30, 0.6/40, 0.2/50).”

Step 1: Represent the semantic statement in technical form:

Probability-distribution (young) is (1/5, 0.9/25, 0.85/30, 0.6/40, 0.2/50)..

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

X isr R, where X is Probability-distribution (young) and R is (1/5 + 0.9/25, 0.85/30 + 0.6/40 + 0.2/50).

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triple components for the GCL statement in 2 can be as follows:

RDF_1:

Subject: *Young*

Predicate: *Probability-distribution*

Object: *_:proba_Bag*

RDF_2:

Subject: *_:proba_Bag*

Predicate: *rdf:type*

Object: *rdf:Bag*

RDF_3:

Subject: *_:proba_Bag*

Predicate: *rdf_1*

Object: *1/5*

RDF_4:

Subject: *_:proba_Bag*

Predicate: *rdf_2*

Object: *0.9/25*

RDF_5:

Subject: *_:proba_Bag*

Predicate: *rdf_3*

Object: *0.85/30*

RDF_6:

Subject: *_:proba_Bag*

Predicate: *rdf_4*

Object: *0.6/40*

RDF_7:

Subject: *_:proba_Bag*

Predicate: *rdf_5*

Object: *0.2/50*

See Figure A.8 for RDF graph.

Figure A.8: RDF Graph Example A.8

- **Fuzzy graph constraint “r: fg”; X isfg R**

Consider the statement: “F* is the Fuzzy graph of F.”

Step 1: Represent the semantic statement in technical form:

Fuzzy-Graph (F) is F*.

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ is } R$, where X is Fuzzy-Graph (F) and R is F*.

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triple components for the GCL statement in 2 can be as follows:

Subject: F

Predicate: *Fuzzy-Graph*

Object: F^*

The subject, predicate, and the object each have a URI that defines it. See figure A.9 for RDF graph.

Figure A.9: RDF Graph Example A.9

- **Usuality constraint “r: u”; X isu R**

Consider the statement: “Usually it’s cold in March.”

Step 1: Represent the semantic statement in technical form:

Probability (Weather (March) is cold) is usually.

Step 2: Represent the technical statement in step 1 in GCL format.

The technical statement in step 2 can be described in GCL form as follows:

$X \text{ is } R$, where X is Usuality (Weather (March) is cold) and R is usually.

Step 3: Represent the GCL statement in step 2 in RDF format.

The corresponding RDF triple components for the GCL statement in 2 can be as follows:

Subject: *Weather (March) is cold*

Predicate: *Usuality*

Object: *usually*

See Figure A.10 for RDF graph.

Figure A.10: RDF Graph Example A.10

- **Pawlak set constraint “r: ps” X isps ($\bar{X}$, $\underline{X}$)**

Consider the statement: “Sam X and Lori Y adopted Chris M and Sue T,” and the following:

X family = {Sam, Mari, Keith, Sara}

Y family = {Lori, Moris, Jon, Keven}

M family = {Chris}

T family = {Sue, Sami}

S1 = {family of Sam and Lori} = {Sam, Lori, Chris, Sue}

Lower approximation = M family

Upper approximation = $X \cup Y \cup M \cup T$ families

Step 1: Represent the above statement in GCL form as follows:

$X \text{ is } R$, where X is Approximation (S1) and R is [lower approximation, upper approximation].

Step 2: Represent the GCL statement in step 1 in RDF format.

The corresponding RDF triples components for the GCL statement in 1 can be as follows:

RDF 1:

Subject: *S1*

Predicate: *Approximation*

Object: *_:approximationS1*

RDF 2:

Subject: *_:approximationS1*

Predicate: *lower*

Object: *Lower approximation*

RDF 3:

Subject: *_:approximationS1*

Predicate: *upper*

Object: *Upper approximation*

S1, Lower, and Upper approximation can be described in a URI, or an RDF bag. See Figure A.11 for RDF graph example.

Figure A.11: RDF Graph Example A.11

Appendix B – “Operator” XML Schema

The following code represents the XML Schema that defines the element “Operator” described in section 5.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Operator">
    <xs:choice>
      <xs:element name="arithmetic_operation">
        <xs:complexType>
          <xs:choice>
            <xs:element name="sum">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="unbounded"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="mult">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="unbounded"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="minus">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="power">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="2"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="round">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="abs">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
      <xs:element name="logarithmic_operation">
        <xs:complexType>
```

```

<xs:choice>
  <xs:element name="exp">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="log">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
        <xs:element name="log_base">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:minInclusive value="2"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="trigonometric_operation">
  <xs:complexType>
    <xs:choice>
      <xs:element name="sin">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg" type="arg"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="cos">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg" type="arg"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="tan">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg" type="arg"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="sec">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg" type="arg"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="cosec">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg" type="arg"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="cotan">
        <xs:complexType>
          <xs:sequence>

```

```

        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="sinh">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="cosh">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="tanh">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="sech">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="cosech">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="cotanh">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="arcsin">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="arccos">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="arctan">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="arg" type="arg"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```

</xs:element>
<xs:element name="arccosh">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arccotan">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arccotanh">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arccosec">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arccosech">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arcsec">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arcsech">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arcsinh">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="arctanh">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="arg" type="arg"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>

```

```

</xs:element>
<xs:element name="condition_operation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="if">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="then">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="else" minOccurs="0">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="arg" type="arg"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="logical_relation" type="logical_relation_Type"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="logical_opertation2" type="logical_relation_Type"/>
</xs:choice>
</xs:complexType>
<xs:complexType name="arg">
  <xs:choice>
    <xs:element name="Constant">
      <xs:complexType>
        <xs:choice>
          <xs:element name="integer_number" type="xs:integer"/>
          <xs:element name="float_number" type="xs:float"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="X" type="xs:string" fixed="X"/>
    <xs:element name="Operator" type="Operator"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="logical_relation_Type">
  <xs:choice>
    <xs:element name="less">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="2"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="greater">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="2"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="less_eq">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="2"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>

```

```
        </xs:complexType>
      </xs:element>
      <xs:element name="greater_eq">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="2"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="equal">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="arg" type="arg" minOccurs="2" maxOccurs="2"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:schema>
```

Appendix C – “Pre_defined_function” XML Schema

The following code represents the XML Schema that defines the element “Pre_defined_function” described in section 5.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:complexType name="pre_defined_function">
    <xs:sequence>
      <xs:element name="Type">
        <xs:complexType>
          <xs:choice>
            <xs:element name="gaussmf">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="parameter" type="parameter"/>
                  <xs:element name="parameter" type="parameter"/>
                </xs:sequence>
                <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="2"/>
                <xs:attribute name="Name" type="xs:string" use="required" fixed="gaussmf"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="trapmf">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="parameter" type="parameter"/>
                  <xs:element name="parameter" type="parameter"/>
                  <xs:element name="parameter" type="parameter"/>
                  <xs:element name="parameter" type="parameter"/>
                </xs:sequence>
                <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="4"/>
                <xs:attribute name="Name" type="xs:string" use="required" fixed="trapmf"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="trimf">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="parameter" type="parameter"/>
                  <xs:element name="parameter" type="parameter"/>
                  <xs:element name="parameter" type="parameter"/>
                </xs:sequence>
                <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="3"/>
                <xs:attribute name="Name" type="xs:string" use="required" fixed="trimf"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="zmf">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="parameter" type="parameter"/>
                  <xs:element name="parameter" type="parameter"/>
                </xs:sequence>
                <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="2"/>
                <xs:attribute name="Name" type="xs:string" use="required" fixed="zmf"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="smf">
              <xs:complexType>
```



```

        <xs:sequence>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
        </xs:sequence>
        <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="2"/>
        <xs:attribute name="Name" type="xs:string" use="required" fixed="smf"/>
    </xs:complexType>
</xs:element>
<xs:element name="dsigmf">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
        </xs:sequence>
        <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="4"/>
        <xs:attribute name="Name" type="xs:string" use="required" fixed="dsigmf"/>
    </xs:complexType>
</xs:element>
<xs:element name="gauss2mf">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
        </xs:sequence>
        <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="4"/>
        <xs:attribute name="Name" type="xs:string" use="required" fixed="tgauss2mf"/>
    </xs:complexType>
</xs:element>
<xs:element name="pimf">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
        </xs:sequence>
        <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="4"/>
        <xs:attribute name="Name" type="xs:string" use="required" fixed="pimf"/>
    </xs:complexType>
</xs:element>
<xs:element name="psigmf">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
        </xs:sequence>
        <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="4"/>
        <xs:attribute name="Name" type="xs:string" use="required" fixed="psigmf"/>
    </xs:complexType>
</xs:element>
<xs:element name="gbellmf">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
            <xs:element name="parameter" type="parameter"/>
        </xs:sequence>
        <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="3"/>
        <xs:attribute name="Name" type="xs:string" use="required" fixed="gbellmf"/>
    </xs:complexType>

```

```

</xs:element>
<xs:element name="sigmf">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="parameter" type="parameter"/>
      <xs:element name="parameter" type="parameter"/>
    </xs:sequence>
    <xs:attribute name="NumOfParam" type="xs:integer" use="required" fixed="2"/>
    <xs:attribute name="Name" type="xs:string" use="required" fixed="sigmf"/>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="Name" type="xs:string" use="optional"/>
<xs:attribute name="LowerRange" type="xs:double" use="optional"/>
<xs:attribute name="UpperRange" type="xs:double" use="optional"/>
<xs:attribute name="LowerDomain" type="xs:double" use="optional"/>
<xs:attribute name="UpperDomain" type="xs:double" use="optional"/>
</xs:complexType>
<xs:complexType name="parameter">
  <xs:sequence>
    <xs:element name="number" type="xs:double"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```