CONTENT-ORIENTED NEGOTIATION IN E-COMMERCE

by

Reyhan Aydoğan

BS, in Computer Engineering, Dokuz Eylül University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2006

ACKNOWLEDGEMENTS

First of all, I thank my thesis supervisor Asst. Prof. Pınar Yolum for her invaluable support during my thesis study as well as her guidance, patience, and motivation.

I thank Prof. Dr. Levent Akın and Prof. Dr. Nadia Erdoğan for being a part of the committee.

I would like to express my gratitude to my family for their endless love, self-sacrifice and support as well as helping me to form my personality. I owe them everything. I want to dedicate this thesis to my dear father, Çetin Aydoğan who is the most valuable person in my life. Special thanks to him. After his death, my mother Emine Aydoğan and my sister Özlem Aydoğan motivated and supported me to complete my graduate education. Without them, I would have never completed this study.

I want to thank all people and friends that I met through my graduate study. I am proud to meet them all. Particularly, I thank my friends Arzucan Özgür, Demet Ayvaz, Esma Kılıç, Hande Zırtıloğlu, Murat Şensoy and Rabun Koşar for their support, motivation and the unique friendship that we share throughout these years.

An earlier version of this work has appeared in the AAMAS Workshop on Business Agents and the Semantic Web. I thank the reviewers of the workshop for their constructive comments. In addition to my parent's endless support, Tübitak has supported me financially during my thesis.

Finally, I would like to thank all my teachers, who have shared their knowledge and ideas with me over the years, and enlightened my path. I thank Prof. Dr. Ethem Alpaydın for sharing his machine learning knowledge with me. I thank my colleagues in both Artificial Intelligent Research Group and Web Services Research Group for their discussions, feedback, and sharing of knowledge about the concepts.

ABSTRACT

CONTENT-ORIENTED NEGOTIATION IN E-COMMERCE

This thesis proposes an approach for automating the content negotiation of services as an alternative to price-oriented negotiation approaches. In content-oriented negotiation, the description of services rather than their price are negotiated. Both consumers and producers share an ontology about the service of interest. The shared ontology contains the features for a given service and captures the relations between them as well as providing the representation of semantics.

Through repetitive interactions, the provider learns consumers' needs accurately and can make better targeted counter offers. In order to learn the consumer preferences, several approaches are explored such as the extension of Version Space, the use of decision trees in an incremental way and the combination of learning with a variety of semantic similarity metrics. The main contributions of this thesis are the extension of Version Space with the capability of learning the disjunctive concepts and a new semantic similarity metric using the taxonomies.

The proposed architecture for the negotiation combines important ideas from incremental learning techniques with the expressive representation of ontologies. In addition to the theory of the automated negotiation architecture, the details of the implemented system are given. A variety of learning algorithms and similarity metrics are tested. The test results show the constructive effect of considering the semantic closeness among services.

ÖZET

ELEKTRONİK TİCARETTE İÇERİK TABANLI PAZARLIK

Bu tez, ücret tabanlı pazarlık yaklaşımlarına alternatif olarak, servis içeriği üzerine yapılan pazarlığın otomatikleştirilmesini temel alan bir yaklaşımı önermektedir. İçerik tabanlı pazarlıkta, ücretten ziyade servisin tanımı üzerine pazarlık yapılmaktadır. Tüketicinin ve üreticinin her ikisi de ilgilendikleri servis hakkındaki bir ontolojiyi paylaşmaktadır. Paylaşılan ontoloji, anlamsal bilgilerinin gösterimini sağladığı gibi, belirlenmiş bir servisin özelliklerini kapsar ve bunlar arasındaki ilişkileri muhafaza eder.

Tekrarlayan etkileşimler sonunda, üretici tüketicinin ihtiyaçlarını doğru olarak öğrenir ve tüketicinin daha çok beğeneceği teklifler üretir. Tüketicinin tercihlerini öğrenmek için Versiyon Uzayı'nın genişletilmesi, karar ağaçlarının artımlı olarak kullanımı ve öğrenmenin çeşitli anlamsal benzerlik ölçütleriyle birleştirilmesi gibi çesitli yaklaşımlar çalışılmıştır. Bu tezin temel katkıları, ayıran kavramları öğrenebilme kabiliyeti ile Versiyon Uzayı'nın genişlemesi ve taksonomileri kullanan yeni bir anlamsal benzerlik ölçütüdür.

Pazarlık için önerilen yapı, artımlı öğrenme tekniklerinden ve ontolojilerin anlamlı gösterimlerinden önemli fikirleri birleştirmektedir. Otomatikleşmiş pazarlık yapısının teorisinine ek olarak, gerçekleştirilen sistemin detayları da verilmektedir. Çeşitli öğrenme algoritmaları ve benzerlik ölçütleri test edilmektedir. Test sonuçları servisler arasındaki anlamsal yakınlığı göze almanın yapıcı bir etkisi olduğunu göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

A	Selected test attribute for decision tree
a	An attribute
$Attributes$	Candidate attributes for being test attribute for the tree
$AVG - SM$	Average similarity
c	A concept or class
c_0	The most specific parent of two concepts ($c1$ and $c2$)
c_1	The first concept whose similarity will be examined
c_2	The second concept whose similarity will be examined
c_h	The number of samples that the hypothesis h covers
$C4.5$	Extended version of ID3 algorithm
$common$	The number of matched attributes
$commonParent$	The most specific common parent of two concepts
d	A training example
$different$	The number of unmatched attributes
$Examples$	Training examples
$Examples_{vi}$	The examples having the value vi for the test attribute
G	The set of most general hypotheses
g	A hypothesis in the most general set
gh	A derivation of an existing general hypothesis
$ID3$	A basic decision tree algorithm
$ID5R$	An incremental decision tree algorithm
m	A constant decreasing the similarity value for parent-child
$MaxSim$	The maximum similarity
n	A constant decreasing the similarity value for siblings
N_0	The number of edges between the parent and root concepts
N_1	The number of edges between the parent and $c1$ concepts
N_2	The number of edges between the parent and $c2$ concept
$P(C)$	The probability of an arbitrary selected object belongs to C
p_i	The proportion of examples belong to class c

R	Rating of the service
$Root$	The root node of the decision tree
S	The set of most specific hypotheses
s	A hypothesis in the most specific hypotheses set
sd	A hypohesis covers both the hypothesis s and the example d
sh	A derivation of an existing specific hypothesis
$S(c_1, c_2)$	The set of concepts that subsumes both c_1 and c_2
SE	Set of training examples
SE_v	Set of training examples having the value v for the attribute
$SM_{(h,service)}$	Similarity of hypothesis h with the given service
$Target\_attribute$	The attribute whose value is to be predicted by the tree
$Values(a)$	Collection of all possible values for the attribute
v_i	i^{th} possible value for the attribute
xi	i^{th} feature
α	Coefficient for the number of common attributes
β	Coefficient for the number of different attributes
CEA	Candidate Elimination Algorithm
DAML	DARPA Agent Markup Language
DT	Decision Tree
MVS	Modified Version Space
OIL	Ontology Inference Layer
OWL	Web Ontology Language
RDF	Resource Description Framework
SCR	Similarity to Current Request
SHOE	Simple HTML Ontology Extensions
SMVS	Similarity to Modified Version Space
SPARQL	Simple Protocol and RDF Query Language
SVS	Similarity to Version Space
URI	Uniform Resource Identifier

VS	Version Space
XML	Extensible Markup Language

1. INTRODUCTION

Traditional e-commerce applications are targeted for human users. However, as the number and extent of transactions increase, there is a tremendous demand for developing e-commerce applications that can be flexibly used by machines. Agents have proven to be a successful paradigm for autonomous and intelligent software such as those needed for flexible e-commerce applications. In a typical e-commerce application, there are two basic but important roles: producer and consumer. Producer advertises and provides a service for which the consumer has interest, whereas the consumer requests and possibly accepts the service. In this case, service can be selling a book, reserving a hotel room, and so on.

Current approaches to e-commerce treat service price as the primary construct for negotiation. However, negotiation on price presupposes that other properties of the service have already been agreed upon. Nevertheless, many times the service provider may not be offering the exact requested service due to lack of resources, constraints in its business policy, and so on. When this is the case, the producer and the consumer need to negotiate the *content* of the requested service [1, 2]. Besides, it is not certain the cheapest service is always the best service. The consumer's needs or interests are primarily the service itself. From this point of view, an approach to negotiation that does not only consider the price but also considers the content of the service necessary.

Most of the current studies related to negotiation assume that the service content is fixed and thus focus on the price of the service [3]. Other approaches assume that all parts of the service are equally important and hence negotiate each service feature one by one [4, 5]. However, usually each service feature is not equally important for a consumer. Moreover, importance of a service feature may vary from consumer to consumer. For instance, completion time of a service may be important for one consumer whereas the quality of the service may be more important for a second consumer.

Without doubt, considering the preferences of the consumer has a positive impact on the negotiation process. For this purpose, evaluation of the components making up the service with different weights can be thought as a well-defined solution. Some studies take these weights as a priori and use the fixed weights [6]. On the other hand, mostly the producer does not know about the consumer's preferences before the negotiation. Thus, if a producer can learn the preferences of a consumer during the interaction, it can provide counter offers that are better targeted for that particular consumer. Accordingly, this thesis develops a content-oriented negotiation framework in which a consumer and a producer agent negotiate on a service. The particular service we have used is a wine selling service. The wine seller learns the wine preferences of the customer to sell better targeted wines.

Learning consumer's preferences by using the information obtained from the dialogues between the producer and consumer requires an incremental learning approach that can be trained at the run time. Contrary to non-incremental learning, in incremental learning the training data are not available at hand before the learning starts. That is data become available incrementally as the participants carry out a dialogue. As far as storage media is concerned, this feature of incremental learning can be thought as a kind of superiority over non-inremental learning. There are several incremental learning algorithms [7] such as Candidate Elimination Algorithm (CEA) [8, 9], ID5R [10] and so on. However, there may be some problems in the case of applying them for learning preferences of the consumer. For example, CEA is known to perform poorly if the information to be learned is disjunctive. Interestingly, consumer preferences are many times disjunctive. Say, we are considering an agent that is buying wine. The consumer may prefer red wine or rose wine but not white wine. This is a disjunctive preference. To use CEA with such preferences, a solid modification is necessary.

Another significant point in negotiation is that, the counter offers for the producer can be refined in time. The question is how the producer uses the information that was learnt from the dialogues to make the best offer to the consumer. Offering the service, which is the most similar service to the learnt target service may be considered as a way of refinement of the counter-offers. Nevertheless, in most cases semantic closeness

is not concerned in similarity estimation between the services and the learnt target service despite the fact that considering semantics may be useful to find the correct service that will be possibly accepted by the consumer. For instance, if the consumer wants to buy a red wine but there is not any red wines in the producer's inventory, then she may be conviced to buy a rose wine more easily rather than to buy a white wine. Taking everything into account, using the semantic similarity would give better results since rose wine is more similar to red wine than white wine.

Moreover, providing a common understanding among different parties (the producer and the consumer agents) is another important issue related to negotiation in e-commerce. To make the most of the communication, two requirements have to be fulfilled. One, the language should be be free from ambuguities. Two, the language should be expressive enough for parties to understand and negotiate details. To achieve both of these requirements, a shared and standard vocabulary is needed. With regard to these matters, the usage of a common ontology, which is widely used in the field of *Semantic Web* [11, 12] comes as a good solution.

The main contribution of this thesis is to propose a multi-issue negotiation mechanism based on the content of the service. This negotiation scheme is designed to be used in e-commerce and it has a different approach from the existing ones, which concerns the price as the primary construct. According to our appoach, the price is only one of the components making up the service and negotiation is performed in respect to all components of the service. This mechanism is fully automated by the use of autonomous agents.

Furthermore, for common understanding of the agents we propose to use a shared ontology as a common vocabulary for the negotiation. In addition to common understanding, the usage of ontology also enables use of the semantics due to the fact that the details of service that is being negotiated can be reasoned. Ontology reasoning is also beneficial to deduce new information from the existing ones. Also, using ontologies enables a negotiation to be independent of the service, so more flexible systems would be established. Whenever the service content is changed, we do not have to

pay attention to change the software related to agents. We only deal with the changes on ontology that is used in the negotiation. Then, everything would be actualized automatically.

Since the preferences of the consumer agent significantly affect the course of action in the negotiation and mostly the provider agent do not have information about these preferences, our negotitation mechanism involves learning preferences. The learning phase is based on the dialogues between the producer and the consumer. By taking the requests as positive examples and the counter-offers rejected by the consumer as negative examples, an incremental learning approach that can be trained at run time and can revise itself with each new example, is applied. Firstly, we propose to use Candidate Elimination Algorithm, which is one of the inductive learning methods for learning the concepts from positive and negative examples. However, it does not support the disjunctive concepts to be learnt. Therefore, we make some modification over this algorithm and make it support disjunctions in a way. One of the significant contributions is the extension of this algorithm to be able to learn disjunctive concepts. Alternatively, we use the decision trees to learn the target service that the consumer wants. In this case, a non-incremental learning algorithm ID3 is applied iteratively. In addition, a comparison of these two algorithms is done.

While the producer is generating counter-offers, the similarity between the available services and the target service that can be acceptable by the consumer is important. According to our automation mechanism, the producer will offer the most similar service to the target service learnt over time during the interaction. Different from other studies, we propose to use the semantic information in similarity estimation. Semantic similarity estimation is performed with the help of the hierarchical information represented in the ontology. Thus, we combine the information learnt by a machine learning technique with the semantics represented in the shared ontology. Another contribution of this thesis is to construct a new semantic similarity metric based on the taxonomy that can be represented in an ontology. Our similarity metric performs as well as and in same scenarios better than the existing ones.

2. BACKGROUND

Before the emergence of the Semantic Web [11, 12], the resources on World Wide Web could only be processed by humans. As the amount of information on the Web increased drastically, it became difficult to find the right piece of information.

By the help of the Semantic Web, the information on the Web becomes both human and machine understandable [13]. The vision of Semantic Web is to understand and process the information on the Web automatically, ideally with software agents. In this case, the Web designer does not only deal with the presentation of information, but also with meaning of its contents and structure. Since the meaning is attached to the statements, machines can also attemp to understand the information on the Web. With this feature, the Web becomes a place providing more advanced and automated service rather than just simple search capabilities.

By attributing semantics to the Web, more advanced search engine can be developed. Current keyword-based search engines return too many irrelevant pages, which takes vast amount of time to find the desired information. More relevant pages can be retrieved by the advanced search engines that can process the information. For instance, a task-centered knowledge support is studied by Jasper and Uschold [14]. According to them, if the system understands the task of users, it may give better services to the user. For instance, when the user searches for the keyword "car" with the intention of repairing a car, the system may perform the search in accordance with the "to repair a car" instead of a general search for the concept "car".

Another study related to Semantic Web is done by Lassila and Adler [15]. They propose to apply Semantic Web onto Ubiquitous Computing in that a semantic gadget in a museum is investigated. The gadget guides and recommends the user in accordance with the environment conditions with using semantics. For example, if the temperature of the museum is too warm and we do not like to carry our coat, the gadget may suggest leaving it in the car.

Possibly, one of the most significant term in Semantic Web is the term of *Ontology*. Ontology is a shared and common understanding of the knowledge concerning the domain of interests, based on conceptualization [16]. It is described by Gruber as "specification of conceptualization" [17]. Shortly, an ontology contains the specification of concepts and their meanings. We describe the concepts, specify their properties and establish some relationships among them by considering a domain of knowledge or interest. It can be thought as representation of knowledge with its semantics. For instance, consider a domain about wine. In that case, the ontology contains the description of a wine concept including its properties such as color, body, winery and so on. In addition to these, the ontology includes some relationships such as "Has-a" and "Is-a". For example, *Chardonnay* is a *Wine* and *Wine* has *color* where color may be one of them: red, rose or white.

Relationships such as "Has-a" and "Is-a" contribute to reasoning of agents. We can represent a taxonomy or hierarchical information by using the relations, "Is-a" and "subclass". By using these relationships, agents can discover new knowledge from the existing knowledge. For instance, using the wine ontology the following reasoning can be made: *Bordeaux* is defined as a *Wine*, *Medoc* is defined as a *Bordeaux* and *Pauillac* is defined as a *Medoc*. When an agent wants to buy wine, an other agent can offer any instance of *Bordeaux*, *Medoc* and *Pauillac* since it can reason that if *Medoc* is a *Bordeaux* and *Bordeaux* is a *Wine* then *Medoc* is a *Wine* and then if *Pauillac* is a *Medoc* and *Medoc* is a *Wine* then *Pauillac* is a wine. Such reasoning makes an agent use the semantic information related to the concepts.

2.1. Ontology Languages

Ontology provides common vocabularies and can be shared among communities. Also, the vocabularies and interpretation of concepts are free from the ambiguity. Each agent processes that ontology can obtain the same information. To express the ontology, we need a language providing formal semantic and syntax. When we classify the languages corresponding to their syntax, there are a variety of markup languages such as SHOE [18], DAML [19], OIL [20], RDF, RDF Schema, OWL and so on.

```
<rdf: RDF>
  <rdf:Description about=‘‘http://www.cmpe.boun.edu.tr/msthesis/RA2006’’>
     <Author rdf:resource=‘‘http://www.cmpe.boun.edu.tr/Reyhan’’/>
  </rdf:Description>
</rdf:RDF>
```

Figure 2.1. Syntax of RDF

From these languages, we explain only RDF, RDF Schema and OWL, which are used in this thesis.

2.1.1. RDF

RDF stands for Resource Decription Framework [21], which is for describing resources on the Web. The resources are identified by the unique names, URI's. RDF is composed of statements which are in a form of triples. A triple consists of subject, predicate and object. The first two are resources and the object may be a resource or a literal such as a string or a float [22]. A predicate establishes binary relations between subjects and objects.

There are several attributes such as *rdf:ID*, *rdf:about*, *rdf:type* and so on. The attribute *rdf:ID* is for identification of resource where *rdf:about* is for referring a resource. *rdf:type* specifies the type of subject.

RDF documents are structured as XML documents. Figure 2.1 indicates an example syntax of RDF. This sample code expresses that the resource http://www.cmpe.boun.edu.tr/msthesis/RA2006 (the subject) has an author (the predicate) http://www.cmpe.boun.edu.tr/Reyhan (the object).

2.1.2. RDF Schema

RDF Schema [23] is language for describing RDF vocabularies. This language can be considered as an extension of RDF. For example, we cannot describe our RDF vocabularies with RDF or we cannot represent a hiearchy of classes. With RDF Schema, we accomplish them in that RDF Schema lets us create our vocabularies and by using "subClassOf" property we also represent the hierarchical organization. With RDF Schema, we can use some constraints on properties such as the domain and the range. The domain represents the type of subject and the range specifies the set of all legal values for the object.

2.1.3. OWL

The expressiveness of RDF and RDF Schema is inadequate for more advanced tasks. We need a more expressive language that provides additional properties and vocabularies. The Web Ontology Language (OWL) improves of RDF and RDF Schema by additional constructs such as advanced properties, contraints, restrictions and so on. These additional features of OWL let us perform more sophisticated reasoning. Expressions in OWL are based on description logics.

We have two types of properties: *object property* and *data property*. The former relates the instances of two objects where the later relates an instance with a data type such as string, float and so on [22]. Particular to object properties, there are some characteristics such as symmetric, transitive, functional, inverseOf and inverse functional properties [24].

Let's look at an example. Figure 2.2 indicates a code snippet in OWL. The vocabularies with the prefix rdfs belongs the RDF Schema. According to the code, there are three OWL classes: *WineGrape*, *WhiteGrape* and *RedGrape*. With "subClassOf", we establish the relation that *WhiteGrape* is a *WineGrape* and *RedGrape* is a *WineGrape*. The hierarchical organization of class is done easily. In the following code, an OWL object property is describes as *hasSugar*. Type of this property is functional

```
<owl:Class rdf:ID="WineGrape" />
<owl:Class rdf:ID="WhiteGrape">
  <rdfs:subClassOf rdf:resource="#WineGrape"/>
</owl:Class>
<owl:Class rdf:ID="RedGrape">
  <rdfs:subClassOf rdf:resource="#WineGrape"/>
</owl:Class

<owl:ObjectProperty rdf:ID="hasSugar">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineSugar" />
</owl:ObjectProperty>
```

Figure 2.2. Syntax of RDF Schema with OWL

property in OWL that wine has only one wine sugar level. This property is also related to another property *hasWineDescriptor* with "'subPropertyOf'. Moreover, this property can only take a value of *WineSugar* since it is specified with the "rdfs:range".

Another example contains a transitive property. Figure 2.3 contains the OWL code in which the class Territory is described and the object property "covers" is declared as transitive. To illustrate this *USRegion* is an instance of *NorthernAmericaCountry* class and *TexasRegion* is an instance of *Territory* class. We give the axioms such as *USRegion* covers *TexasRegion* and *TexasRegion* covers *EdnaValleyRegion*. When we reason the ontology, we can also infer the fact that *USRegion* covers *EdnaValleyRegion*. As it seen, these features advance the reasoning and extracting new axioms from the existing ones are enabled.

2.2. Query Language

We add axioms as RDF statements consisting RDF triples (subject, predicate, object). To obtain the information from these requires a query language. Simple Pro-

```
<owl:Class rdf:ID="Territory"/>
<owl:ObjectProperty rdf:ID="covers">
   <rdf:type rdf:resource="&owl;TransitiveProperty" />
   <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
   <rdfs:range rdf:resource="#Territory" />
 </owl:ObjectProperty>
<NorthernAmericaCountry rdf:ID="USRegion">
   <covers rdf:resource="#TexasRegion" />  ...
</NorthernAmericaCountry>

<Territory rdf:ID="TexasRegion">
   <covers rdf:resource="#EdnaValleyRegion" />  ...
</Territory>
```

Figure 2.3. Syntax of OWL

tocol and RDF Query Language (SPARQL) [25] is one of the query languages for RDF graph. It includes "Select" and "where" clauses. In "Select" clause, we put the variables being returned in result set and some triple patterns as a condition is replaced in "where" clause. Figure 2.4 represents a query written in SPARQL. Consider our RDF documents includes a triple such as ("http://www.cmpe.boun.edu.tr/people/reyhan", "http://www.cmpe.boun.edu.tr/terms/advisor", "http://www.cmpe.boun.edu.tr/ people/pinar"). After running the query, the variable *advisor* will be bounded with http://www.cmpe.boun.edu.tr/people/pinar. SPARQL is supported by the ontology reasoner, JENA. Therefore, we make our queries with SPARQL by using JENA.

2.3. Learning Algorithms

As we have pointed out in the previous chapter, we need an incremental algorithm to learn the preferences. Incremental Learning is the learning that the training examples do not have to exist at the beginning of the learning process. It allows training examples become available over the time [7].

```
SELECT ?advisor
WHERE
{
  <http://www.cmpe.boun.edu.tr/people/reyhan>
  <http://www.cmpe.boun.edu.tr/terms/advisor>
   ?advisor .
}
-----------------------------------------------
Possible result set is:
<http://www.cmpe.boun.edu.tr/people/pinar>
```

Figure 2.4. Syntax of SPARQL

Moreover, all past samples are not required to be stored. Most incremental algorithms came in the form of inductive learning.

Inductive Learning is a learning technique, which is based on the observed data. From the observed samples, we try to deduce some general rules. Firstly, we classify the samples by labelling each input with a class name. Then, we construct classification over the dataset. This process is called concept learning or induction [26]. In this thesis, we first study one of inductive learning methods, Version Space.

Next, we study decision trees [9, 27], which are widely used structure for supervised learning. We build decision trees iteratively during dialogues so that they fulfill the requirements of incremental learning algorithms. We review Version Space and Decision Trees next.

2.3.1. Version Space

Version Space [8] is one of the inductive learning approaches that learns concepts from observed examples. The aim of version space algorithm is to generate general rules or hypotheses, which involve the positive examples and do not include any negative

Algorithm 1 Candidate Elimination Algorithm

1: $G \leftarrow$the set of maximally general hypotheses in H
2: $S \leftarrow$the set of maximally specific hypotheses in H
3: For each training example, d
4: **if** d is a positive example **then**
5: Remove all hypotheses in G do not cover d
6: For each hypothesis s in S does not cover d
7: – Remove s from S
8: – Add to S all minimal generalizations sh of s such that sh covers d and some members of G are more general than sh
9: – Remove from S any hypothesis that is more general than another hypothesis in S
10: **end if**
11: **if** d is a negative example **then**
12: Remove all hypotheses in S that cover d
13: For each hypothesis g in G does cover d
14: – Remove g from G
15: – Add to G all minimal specifications gh of g such that gh does not cover d and some members of S are more specific than gh
16: – Remove from G any hypothesis that is less general than another hypothesis in G
17: **end if**

Figure 2.5. Candidate Elimination Algorithm

examples. In fact, this general hypothesis represents the desired concept. To understand how the Version Space approach works, we investigate the Candidate Elimination Algorithm (CEA) that is one of the incremantl learning algorithm implementing the Version Space. Figure 2.5 gives the pseudo code [9].

There are two kinds of sets representing the most general hypotheses (set G) and the most specific hypotheses (set S). These sets are used to capture the limits of the learned concept. That is, a concept is at least as specific as the hypothesis that exists in set S and at most as general as the hypothesis contained in set G. The boundaries

of the most general hypotheses are as large as possible in that they cover the entire positive training example and possible positive instances; whereas those of the most specific hypotheses are as small as possible they minimally cover the observed positive samples. By using the most general hypotheses and the most specific hypotheses, we can produce all possible hypotheses for the target concept. As the algorithm receives positive and negative examples, it revises the G and S such that eventually G and S intersect in which case the concept has been learned.

At the beginning of the algorithm, G is initialized with the set of maximally general hypotheses that cover everything, whereas S is set with the set of maximally specific ones (lines 1–2). We classify the examples as positive and negative. Then, we modify the hypotheses set by examining the class of the coming example. The algorithm checks whether the coming example is positive (line 4). If the sample is positive, it should be covered by both set G and S. According to Version Space, each hypothesis in S should cover the positive samples. If it does not cover, the algorithm makes it cover by modifying the contents of the hypothesis. The hypothesis not covering the positive sample is eliminated from the G. In addition to this, it removes the hypothesis in S that does not cover the positive sample and adds the minimally generalized version of that removed hypothesis to S, which covers the positive sample now (lines 6–8).

Since the set S consists of the most specific hypotheses and the set G consists of the most general hypotheses, any hypothesis in S is not allowed to be more general than any hypothesis in G. Conversely, any hypothesis in G is not allowed to be more specific than any hypothesis in S. In addition, any hypothesis that is more general than other hypotheses in S is removed (line 9).

If the example is a negative one, it should not be covered by any hypothesis in either G and S. If the most specific set contains any hypothesis covering that sample, it is eliminated from the S (line 12). The hypotheses in G that cover the negative example is minimally specialized in that no more cover that example (lines 13-15). Furthermore, we eliminate the hypotheses that is more specific than those of the most general set (line 16).

To sum up, by specialization of the most general set and by generalization of the most specific set, both sets become more closer to each other over time. Finally, they will hopefully be equal to each other when the target concept is learnt.

2.3.2. Decision Trees

Decision Tree is one of the most widely used machine learning techniques. We can exract if-then rules that can be understood by humans from the constructed tree. There are several algorithms in generation of decision trees such as ID3 [28], C4.5 [29] that is the extension of ID3. We will investigate the basic decision tree algorithm ID3 developed by Quilan.

Examples are represented by attribute-value pairs and by using a heuristic such as information gain, we try to divide the dataset in homogeneous parts according to the attributes. The selection of test attribute is the most significant task. When a test attribute is selected, the node will be split into several parts corresponding the number of possible values for that attibute. For example, if the test attribute is color and the valid values for the color attribute are red, rose and white, the node related to color is split into three parts. Spliting process will continue until the subset of all nodes are approximately homogeneous.

The algorithm in Figure 2.6 specified in [9] shows how the ID3 Algorithm constructs the decision tree. The information gain will be explained in the following chapter in details but shortly it is an indicator of the homogeneous distribution of examples. It contains entropy estimation for the values of attributes. The attibute whose information gain is the highest is selected as a test attribute. According to the algorithm, if all examples in the set belongs to same class initially, the root node is returned with the label of that class (lines 2–7). We have two classess here: positive and negative. Then, the algorithm checks whether any attribute remains in the *Attributes*. If there is no attibute to test, the root node will be the label of the class that most of the examples belong to (lines 8–10).

Algorithm 2 ID3(*Examples*, *Target_attribute*, *Attributes*)

Require: *Examples* are the training examples. *Targetattribute* is the attribute whose value is to be predicted by the tree. *Attributes* is a list of other attributes that may be tested by the learned decision tree. Returns a decsion tree that correctly classifies the given Examples.

```
Create a Root node for the tree
if all Examples are positive then
    Return the single-node tree Root. with label=+
end if
if all Examples are negative then
    Return the single-node tree Root. with label=-
end if
if Attributes is empty then
    Return the single-node tree Root, with label= most common value of
    Target_attribute in Examples
else
    A ← theattributefromAttributes that best classifies Examples
    {The best attribute is the one with highest information gain}
    The decision attribute for Root ← A
    For each possible value, v_i of A
    Add a new tree branch below Root, corresponding to the test A = v_i
    Let Examples_vi be the subset of Examples that have value v_i for A
    if Examples_vi is empty then
        Below this new branch add a leaf node with label= most common value of
        Target_attribute in Examples
    else
        ID3 Algorithm (Examples, Target_attribute, Attributes − {A})
    end if
end if
Return Root
```

Figure 2.6. ID3 Algorithm

The most significant part of the algorithm is to select the test attribute via the information gain. The algorithm chooses an test attribute from all possible attibutes whose information gain is the highest (line 11). Then, the node is divided into branches with all possible values of the selected test attribute (lines 13-14). If there is no example having the value of that attribute, the newly branched node is accepted as a leaf node with the most common value for the class (line 15). Otherwise, the algorithm performs the same process for the new nodes branched to the current node recursively (line 19).

2.4. Related Work

We review the recent literature in comparison to our work. Debenham [30] studies the management of the single-issue and multi-issue negotiation processes using the market data and information over the Internet. Each transaction such as request and offer is considered as a business process and have some constraints such as time. A negotiation should be completed up to a certain time. These constraints are managed by a multiagent system.

Tama *et al.* [31] propose a new approach based on ontology for negotiation. According to their approach, the negotiation protocols used in e-commerce can be modeled as ontologies. Thus, the agents can perform negotiation protocol by using this shared ontology without the need of being hard coded of negotiation protocol details. This shared ontology provides the vocabulary of the negotiation process for the agents, which can behave dynamically. However, Tama *et al.* do not provide a method for learning preferences during negotiation as we have done in this thesis.

Ontology can be used in service discovery for better results than that based on syntactic matching. Broens *et al.* [32] propose a context-aware, ontology-based service discovery. Matching capabilities can be increased by using the semantic information. For instance, when the user requests a service for selling music, selling CD will be valid since the CD is a subtype of music. They classify the services according to their properties and model the service and service types in an ontology. To find a service, they consider contextual information in addition to inputs and outputs.

Sadri *et al.* [33] study resource allocation problem, which needs the negotiation mechanism. Agents have limited resources and they sometimes require taking missing resources from other agents. A mechanism which is based on dialogue sequences among agents is proposed as a solution. They rely on observe-think-action agent cycle. These dialogues include offering resources, resource exchanges and offering alternative resource. The main idea in this system is that each agent has a goal to accomplish and they have a planner in order to reach their goal. The planner determines the activities. In accordance with the agent's intention, the dialogues are generated automatically to obtain the missing resources. In their study, an ontology has not been used and semantics is not a concern.

Auction mechanism is used in both negotiation and resource allocation. One of the studies about auctions and agents is that of Brandt and Weiβ [34], in which the Vickrey Auction mechanism and the effects of antisocial attitudes of agents over this auction process are examined. In Vickrey Auction, the bidding owning the highest price wins the auction and the second highest price is paid to the winner. Each agent has its own private value of bid, which is not known by other agents and the payoff of the agent is the difference between second highest price and the value of its own bid.

The aim of antisocial agents is to damage its opponent rather than to maximize its own profit. At most cases, the loss of its opponent's profit is more important than their own profit. According to the authors, Vickrey Auction is not suitable for antisocial agents. However, antisocial behavior is inevitable in real world. By considering antisocial attitudes, convenient Vickrey Auction strategies are proposed by Brandt and Weiβ. There are several strategies to predict other agents' private values of bids and give the best price. However, if more than one agent use such strategies, it may be dangerous. These strategies are risky since the agent may lose too much. Their study considers only price whereas our study concerns all of the component making up the service.

Brzostowski and Kowalczyk propose an approach in order to select an appropriate negotiation partner by investigating previous multi-attribute negotiations [35]. For

achieving this, they use case-based reasoning. They emphasize the positive effect of finding the right partner. Their approach is probabilistic since the behavior of the partners can be changed by the next time. In our approach, we are interested in negotiation the content of the service. After the consumer and producer agree on the service, price-oriented negotiation mechanisms can be used to agree on the price.

Fatima *et al.* study the factors, which affect the negotiation such as preferences, reservation time, deadline, price and so on, since the agent who develops a strategy against its opponent should consider all of them [4]. However, obtaining full information about its opponent does not seem possible. The study emphasizes the negotiation parameters such as time and price. These factors may seem independent but time has a strong impact over price. They point out that a negotiation process should be completed in a particular time. The goal of the seller agent is to sell the good or service with high price as possible whereas that of the buyer agent is to buy the good with low price as possible. Time interval affects these agents differently. For instance, late time may lose the utility for buyer agent in case of not having services whereas the seller agent prefers to give the service as late as possible.

Furhtermore, the information of opponent agents such as deadline time and price are incomplete information for agents. Fatima *et al.* use probabilistic distributions over these parameters and construct a model, which shows the utility for each agent in a particular time period and price interval. There are four possibilities. Firstly, both agents can lose utility after a certain time period. Secondly both agents may gain utility. Thirdly, buyer agent can gain utility whereas the seller agent loses. Finally, the seller agents can gain utility whereas the buyer loses. In accordance with this model, the agents should choose the optimal strategy. This study similarly to our study tries to obtain incomplete information. Compared to Fatima *et al.* our focus is different while their focus on the effect of time on negotiation, our focus on learning preferences on a successful negotiation.

Faratin *et al.* propose a multi-issue negotiation mechanism based on trade-offs [6]. The service variables for the negotiation process such as price, quality of the ser-

vice, delivery time and so on are considered traded-offs against each other (i.e., higher price for earlier delivery). They generate a heuristic model for trade-offs including fuzzy similarity estimation and a hill-climbing exploration for possibly acceptable offers. Although we address a similar problem, our main target is to learn the preference of the customer by the help of inductive learning and generate counter-offers in accordance with these learned preferences. Faratin *et al.* only use the last offer made by the consumer in calculating the similarity for choosing counter offer. Unlike them, we also take into account the previous requests of the consumer using Modified Version Space or Decision Tree. In their experiments, Faratin *et al.* assume that the weights for service variables are fixed a priori. On the contrary, we aim to learn these preferences through inductive learning.

3. PROPOSED NEGOTIATION ARCHITECTURE

Our main components are consumer and producer agents, which communicate with each other to perform content-oriented negotiation. To make the most of the communication, two requirements have to be fulfilled. One, the language the agents speak should be free from ambiguities. Two, the language should be expressive enough for parties to understand and negotiate details. To achieve both of these requirements, we use ontologies in our architecture.

The architecture contains two types of ontology with different purposes: a shared ontology for common understanding and a specific ontology for the producer agent with the intention of a data repository. A shared ontology provides the necessary vocabulary for agents and hence enables a common language for agents. This ontology describes the content of the service. Further, since an ontology can represent concepts, their properties and their relationships semantically, the agents can reason the details of the service that is being negotiated. Since a service can be anything such as selling a car, reserving a hotel room, and so on, the architecture is independent of the ontology used. However, to make our discussion concrete, we use the well-known Wine ontology [36] with some modification to illustrate our ideas and to test our system.

The wine ontology describes different types of wine and includes features such as color, body, winery of the wine and so on. With this ontology, the service that is being negotiated between the consumer and the producer is that of selling wine. The specific ontology of the producer agent holds the inventory information of the producer. This ontology comprises which products the producer owns, the number of the products and ratings of those products. Ratings indicate the popularity of the products among customers. Those are used to decide which product will be offered when there exists more than one product having same similarity to the request of the consumer agent.

Figure 3.1 depicts our architecture. The consumer agent represents the customer and hence has access to the preferences of the customer. The consumer agent generates

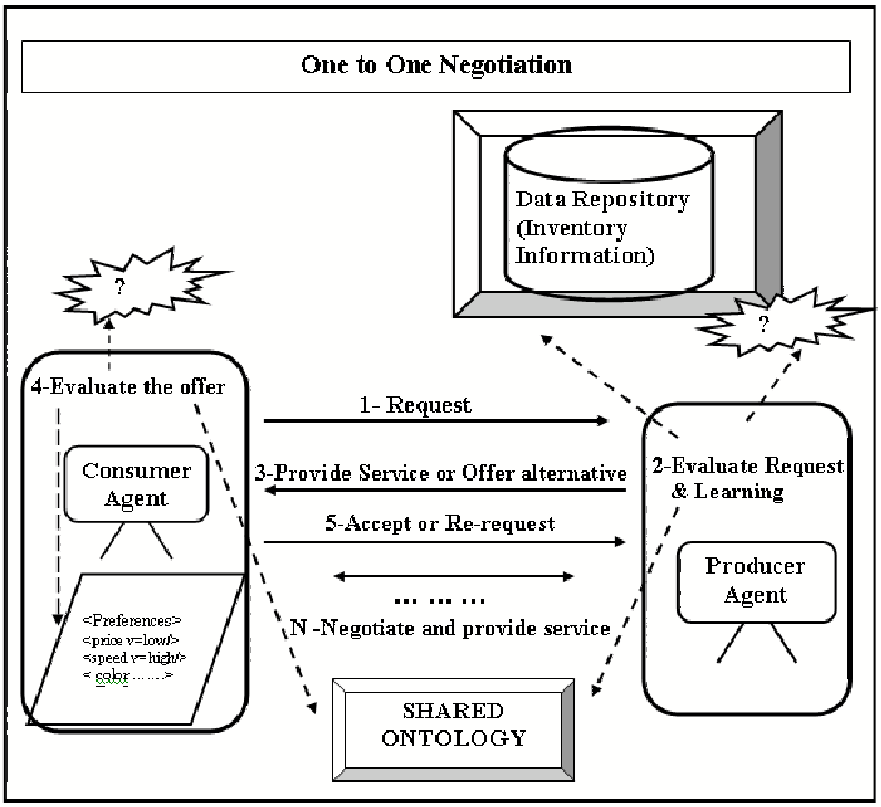

Figure 3.1. Negotiation architecture

request in accordance with these preferences and negotiates with the producer based on these preferences. Similarly, the producer agent has access to the producer's inventory and knows which wines are available or not. The inventory information is represented as a set of individual classes in the ontology.

The architecture advances in a turn-taking fashion, where the consumer agent starts the negotiation with a particular service request. The request is composed of significant features of the service. In the wine example, these features include color, winery and so on. This is the particular wine that the customer is interested in purchasing. If the producer has the requested wine in its inventory, the negotiation ends. Otherwise, the producer offers an alternative wine from the inventory. Moreover, the process of offering an alternative wine involves some learning phase and evaluation of the best offer in accordance of the similarity to the request. When the consumer receives a counter offer from the producer, it will evaluate it. If it is acceptable, then the negotiation will end. Otherwise, the customer will generate a new request or stick to the previous request. This process will continue until some service is accepted by the

consumer agent or all possible offers are put forward to the consumer by the producer.

One of the crucial challenges of the content-oriented negotiation is the automatic generation of counter offers by the service producer. When the producer constructs its offer, it should consider three important things: the current request, consumer preferences and the producer's available services. Both the consumer's current request and the producer's own available services are accessible by the producer. However, the consumer's preferences in most cases will not be. Hence, the producer will have to understand the needs of the consumer from their interactions and generate a counter offer that is likely to be accepted by the consumer. This challenge can be studied in four stages:

- Representation: How do we represent the requests and counter offers of the producer so that they can reason about each other's actions? (Section 3.1)
- Learning: How can the producers learn about each customer's preferences based on requests and counter offers? (Section 3.2)
- Similarity Estimation: How can the producer agent estimate similarity between the request and available services? (Section 3.3)
- Revision: How do the agents revise their requests or offers based on incoming information? (Section 3.4)

3.1. Representation

The requests of the consumer and the counter offers of the producer are represented as vectors, where each element in the vector corresponds to the value of a feature. For instance, the customer may prefer medium and strong red wine. This can be represented with the following vector query *(Medium, Strong, Red)*. Or, if the customer prefers light and delicate white wine, this can be represented with the following query *(Light, Delicate, White)*.

3.2. Learning Phase

At the beginning, the producer agent does not know about consumer's preferences but the producer should learn those preferences by the time in order to serve up better offers. Thus, the producer agent tries to learn consumer's preferences using information obtained from the dialogues between the producer and the consumer. The preferences denote the relative importance degree of the features of the services demanded by the consumer agents. For instance, the color of the wine may be so important that the consumer insists on buying the wine whose color is "red" and rejects all the offers involving the wine whose color is "white" or "rose". On the contrary, the winery may not be as important as the color for this customer, so the consumer may have a tendency to accept any offer, which meets the color requirement, whose winery is different from the specified one in the request.

To tackle this problem, we propose to use inductive learning. This technique is applied to learn the preferences as concepts. There are some alternative learning methods in this area, which are convenient for this task such as Version Space, Decision Trees and so on. We elaborate on Candidate Elimination Algorithm for Version Space, the modified version of this algorithm and ID3 algorithm for decision trees. Then, we compare their performance.

3.2.1. Version Space

Version Space [8] is one of the inductive learning approaches that learns concepts from observed examples. As Section 2.3.1 explains, there are two important sets: the most general set G and the most specific set S. The aim is to obtain the hypothesis containing the target concept. When these two sets are equal to each other, it is accepted that the target concept is learnt. Candidate Elimination Algorithm (CEA) that is also an incremental learning algorithm being able to learn the examples becoming available over time is studied in this part. Figure 2.5 in Section 2.3.1 gives the pseudo code [9]. CEA implements the Version Space.

At the beginning of the algorithm, G is initialized with the set of maximally general hypotheses that cover everything, whereas S contains the set of maximally specific ones. During the interactions, each request of the consumer can be considered as a positive example and each counter offer generated by the producer and rejected by the consumer agent can be thought of as a negative example. At each dialogue between the producer and the consumer, we apply training over sets of the most specific and the most general hypotheses.

Assume that our target concept, wine, has three attributes: body, flavor and color. In fact, Wine class has seven attributes in our implementation. In order to keep the examples simple, here we only include three attributes. According to the scenario, assume a wine with features *(Full, Strong, White)* is an acceptable wine. Initially, G is simply initialized with the most general hypothesis, *(?, ?, ?)* where *?* means this attribute can take any value. On the other hand, S is initialized with the most specific hypothesis consistent with the first positive observed training sample, in this case *(Full, Strong, White)*. Table 3.1 shows the content of the G and S.

Table 3.1. When Candidate Elimination Algorithm fails

Type	**Sample**	**The most general set**	**The most specific set**
+	(Full,Strong,White)	{(?,?,?)}	{(Full,Strong,White)}
-	(Full,Delicate,Rose)	{(?,Strong,?),(?,?,White)}	{(Full,Strong,White)}
+	(Medium,Moderate,Red)	{ }	{(?,?,?)}

The negative samples enforce the specialization of some hypotheses so that G does not cover any hypothesis accepting the negative samples as positive. Following the previous example, assume that *(Full, Delicate, Rose)* is an unacceptable wine (i.e., a negative training sample). The general set should be specialized in such a way that it should not cover this instance but covers previously observed positive samples.

In the algorithm, there may be various ways to specialize the general set G. One way is to compare the attributes of the negative example with the specific set and

to assign their difference to the existing G. Also the algorithm should eliminate the hypothese that cover the negative sample from S.

Assume that G contains *(?, ?, ?)* and S contains *(Full, Strong, White)*. If the negative example *(Full, Delicate, Rose)* comes in, then G needs to be specialized such that it becomes *(?, Strong, ?)* and *(?, ?, White)*.

When a positive sample comes, the most specific set S should be generalized in order to cover the new training instance. Similarly, the details of the generalization is not specified in CEA; but from the illustration explained in [9], it can be performed by replacing the different values between special set and current positive sample with *?*. Further, the most general hypotheses in G that do not cover the new positive sample are eliminated.

Assume that S contains *(Full, Delicate, Rose)*. If the next positive example is *(Medium, Delicate, Rose)*, then the generalization of specific set S will be *(?, Delicate, Rose)*.

As a result, the most general hypotheses and the most special hypotheses cover all positive training samples but do not cover any negative ones. Incrementally, G specializes and S generalizes until G and S are equal to each other. When these sets are equal, the algorithm converges by means of reaching the target concept.

Unfortunately, Candidate Elimination Algorithm is primarily targeted for conjunctive concepts. On the other hand, we need to learn disjunctive concepts in the negotiation of a service. For instance, a consumer may want strong white wine or moderate red wine but not moderate white or strong red wine. In Table 3.1, the behavior of the algorithm is shown clearly in accordance with the given samples. It can be seen that the algorithm fails in learning concepts where disjunctive requests exist. In detail, the general set is specialized in such a way that it no longer contains any hypothesis whereas the special set is over generalized and starts to contain all possible hypothesis. This means that S and G can never intersect. However, learning from such

concepts is required when providing service to the consumer since consumer may have several alternative wishes.

The way the general set G is specialized and the special set S is generalized is crucial for successfully learning the hypothesis. Consider the following example shown in Table 3.2.

Table 3.2. Generalization in Candidate Elimination Algorithm

Type	**Sample**	**The most general set**	**The most specific set**
+	(Light,Delicate,Red)	{(?,?,?)}	{(Light,Delicate,Red)}
+	(Light,Delicate,White)	{(?,?,?)}	{(Light,Delicate,?)}

The color feature of the wine product is allowed to take three different values: white, red and rose. Assume the first positive sample is *(Light, Delicate, Red)*, denoting a red, light and delicate wine. And, the second positive sample is *(Light, Delicate, White)*, denoting a white, light and delicate wine. According to algorithm, the special set S will be generalized to be *(Light, Delicate, ?)*. However, now S also contains samples for *(Light, Delicate, Rose)* even though the consumer has not shown any interest in rose wine so far. If the consumer now rejects an offer of rose, light and delicate wine, the special set will be in violation with the consumer's preferences and the algorithm will end up learning the preferences incorrectly.

To sum up, Candidate Elimination Algorithm works well when the target preferences are in a conjunctive form; on the other hand, this algorithm does not work when the consumer agent has disjunctive preferences. In that case, the system would not learn the preferences so the producer would generate incompatible counter offers or it would not be able to give any counter offer. As a result, the negotitation process may fail in case of that the consumer's preferences are disjunctive.

3.2.2. Modified Version Space

We deal with the problem of not supporting disjunctive concepts of CEA by extending our hypothesis language to include disjunctive hypothesis in addition to the conjunctives and negation. Each attribute of the hypothesis has two parts: inclusive list, which holds the list of valid values for that attribute and exclusive list, which is the list of values which cannot be taken for that feature. For instance, if the most specific set is {*(Light, Delicate, Red)*}, *it will be* {*(Light, Delicate, [White, Red])*} instead of *(Light, Delicate, ?)* after a positive example, *(Light, Delicate, White)* comes. Only when all the values exist in the list, they will be replaced by *?*. In other words, we let the algorithm generalize more slowly than before.

We modify the CEA algorithm in Section 2.3.1 to deal with this change. The modified algorithm is given as Figure 3.2. The initialization phase is the same as the original one (lines 1, 2). If any positive sample comes, we add the sample to the special set as before (line 4). However, we do not eliminate the hypotheses in G that do not cover this sample since G now contains a disjunction of many hypotheses, some of which will be conflicting with each other. Removing a specific hypothesis from G will result in loss of information, since other hypotheses are not guaranteed to cover it. After some time, some hypotheses in S can be merged and can construct one hypothesis (lines 6, 7).

When a negative sample comes, we do not change S as before. We only make the most general hypotheses not cover this negative sample (lines 11–15). Different from the original Candidate Elimination Algorithm, we try to specialize the G minimally. The algorithm removes the hypothesis covering the negative sample (line 13). Then, we generate new hypotheses as the number of all possible attributes by using the removed hypothesis.

For each attribute in the negative sample, we add one of them at each time to the exclusive list of the removed hypothesis. Thus, all possible hypotheses that do not cover the negative sample are generated (line 14). Note that, exclusive list contains

Algorithm 3 Modified Candidate Elimination Algorithm

1: $G \leftarrow$the set of maximally general hypotheses in H
2: $S \leftarrow$the set of maximally specific hypotheses in H
3: For each training example, d
4: **if** d is a positive example **then**
5: Add d to S
6: **if** s in S can be combined with d to make one element **then**
7: Combine s and d into sd {sd is the rule covers s and d}
8: **end if**
9: **end if**
10: **if** d is a negative example **then**
11: For each hypothesis g in G does cover d
12: * Assume : g = (x1, x2, ..., xn) and d = (d1, d2, ..., dn)
13: – Remove g from G
14: – Add hypotheses g1, g2, gn where g1= (x1-d1, x2,..., xn), g2= (x1, x2-d2,..., xn),..., and gn= (x1, x2,..., xn-dn)
15: – Remove from G any hypothesis that is less general than another hypothesis in G
16: **end if**

Figure 3.2. Modified Candidate Elimination Algorithm

the values that the attribute cannot take. For example, consider the color attribute. If a hypothesis includes red in its exlusive list and *?* in its inclusive list, this means that color may take any value except red.

Table 3.3 illustrates an example that works with the modified version space algorithm. The initial request of the consumer is the same as the one in Table 3.1, so as the initial G and S. When the counter offer from the producer comes (negative instance), we specialize G since G should not cover any negative instances. We replace *(?, ?, ?)* by three disjunctive hypotheses; each hypothesis being minimally specialized. In this process, at each time one attribute value of negative sample is applied to the hypothesis in the general set. Again here, we let the algorithm specialize slowly compared to the original algorithm.

Note that in Table 3.3, we do not eliminate {*(?-Full), ?, ?*} from the general set while having a positive sample such as *(Full, Strong, White)*. This stems from the possibility of using this rule in the generation of other hypotheses. For instance, if the example continues with a negative sample *(Full, Strong, Red)*, we can specialize the previous rule such as {*(?-Full), ?, (?-Red)*}. By algorithm in Figure 3.2, we do not miss any information.

Table 3.3. How Modified CEA works

Type	**Sample**	**The most general set**	**The most specific set**
+	(Full,Strong,White)	{(?, ?, ?)}	{(Full,Strong,White)}
-	(Full,Delicate,Rose)	{{(?-Full), ?, ? }, {?, (?-Delicate), ?}, {?, ?, (?-Rose)}}	{(Full,Strong,White)}
+	(Medium,Moderate,Red)	{{(?-Full), ?, ?}, {?,(?-Delicate), ?}, {?, ?, (?-Rose)}}	{{(Full,Strong,White)}, {(Medium,Moderate,Red)}}

3.2.3. ID3

ID3 is an inductive learning algorithm used in constructing decision trees in a top-down fashion from the observed examples represented in a vector with attribute-value pairs. Applying this algorithm to our system with the intention of learning the consumer's preferences is appropriate since this algorithm also supports learning disjunctive concepts in addition to conjunctive concepts.

The ID3 algorithm is used in the learning process with the purpose of classification of offers. There are two classes: positive and negative. Positive means that the service description will possibly be accepted by the consumer agent whereas the negative implies that it will potentially be rejected by the consumer. Consumer's requests are considered as positive training examples and all rejected counter-offers are thought

as negative ones.

The decision tree has two types of nodes: leaf node in which the class labels of the instances are held and non-leaf nodes in which test attributes are held. The test attribute in a non-leaf node is one of the attributes making up the service description. For instance, body, flavor, color and so on are potential test attributes for wine service. When we want to find whether the given service description is acceptable or not, we start searching from the root node by examining the value of test attributes until reaching a leaf node keeping the class type.

Selection of test attributes, which are used to divide the examples into subsets by considering the (im)purity of examples in each group, is a crucial task in construction phase of the tree since it affects the size of tree. Our aim is to build the smallest decision tree. There are several heuristics or goodness functions to find the best test attribute such as information gain, information gain ratio and gini index. ID3 uses information gain, which is a popular criterion for impurity test.

In detail, information gain is the difference of information before the split with that after the split and its value increases with the average purity of subsets. Therefore, we choose the test attribute whose information gain is higher than that of others. In estimation of information gain, an (im)purity measure called entropy represents the homogenity of examples in subsets after the split process.

After selection of a test attribute, tree splits in accordance with all possible values of that attribute. For instance, if the best attribute is "color" in our wine example, the node will be branched into three parts for the values: red, rose and white. This operation will be performed for all new nodes recursively. The splitting will be finished when each subset is homogeneous meaning to have the same class label or to have no attribute remains for testing.

The entropy of a set of examples SE is shown in Equation (3.1) [9] where c represents the class label (positive and negative) and the p_i is the proportion of SE

belong to c. The entropy can take values between zero and one. If all the examples in the SE belong to same class, the entropy will be zero.

$$Entropy(SE) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{3.1}$$

The information gain of the set of examples SE for an attribute a is defined in Equation (3.2) [9] where $Values(a)$ are the collection of all possible values for the attribute a and the ratio $\frac{|SE_v|}{|SE|}$ is the proportion of examples having the value v for the attribute a.

$$Gain(SE, a) = Entropy(SE) - \sum_{v \in Values(a)} \frac{|SE_v|}{|SE|} Entropy(SE_v) \tag{3.2}$$

Let's assume that our target concept, wine, has three attributes: body, flavor and color as in Section 3.2.1. The body can take three values: medium, full and light while flavor can be one of moderate, strong and delicate. Moreover, red, rose and white are possible values for the color attribute. Our sample training examples are listed in Table 3.4.

Table 3.4. Sample training examples of ID3

ID	**Example(body, flavor, color)**	**Class (+ or -)**
E1	(Full, Strong, Red)	Acceptable (+)
E2	(Full, Delicate, Rose)	Not Acceptable (-)
E3	(Medium, Strong, Red)	Acceptable (+)
E4	(Light, Moderate, Red)	Not Acceptable (-)
E5	(Light, Moderate, Rose)	Acceptable (+)
E6	(Medium, Strong, Rose)	Not Acceptable (-)

In our sample set, none of the examples involves white color; therefore, the possible values for color attribute are only red and rose. According to the algorithm, the

information gain is calculated for each attribute to find the best attribute splitting the dataset more homogeneously. Here, the level of homogeneity decreases with the increasing number of samples whose class labels are different from others. The entropies and information gains estimated for the root node are given in Table 3.5. According to algorihtm, 0 $log_2 0$ is equal to zero in calculation of the entropies. Among three attributes, the gain of attribute body is 0.0, that of attribute flavor is 0.20 and that of attribute color is 0.08. The results show that the attribute flavor has the highest information gain. As a result, the flavor attribute is selected as test attributes for the root node and the root splits into three nodes for the values of the attribute flavor: strong, moderate and delicate.

Table 3.5. Entropies for wine attributes of the entire dataset [E1-E6]

Entropies & Information Gains
$Entropy(SE) = -3/6 * \log_2 3/6 - 3/6 * log_2 3/6 = 1$
$Entropy(Body = Light) = -1/2 * log_2 1/2 - 1/2 * log_2 1/2 = 1$
$Entropy(Body = Medium) = -1/2 * \log_2 1/2 - 1/2 * \log_2 1/2 = 1$
$Entropy(Body = Full) = -1/2 * \log_2 1/2 - 1/2 \log_2 * 1/2 = 1$
$Gain(SE, Body) = 1 - 2/6 - 2/6 - 2/6 = 0$
$Entropy(Flavor = Strong) = -2/3 * \log_2 2/3 - 1/3 * \log_2 1/3 = 0.91830$
$Entropy(Flavor = Moderate) = -1/2 * \log_2 1/2 - 1/2 * \log_2 1/2 = 1$
$Entropy(Flavor = Delicate) = -0/1 * \log_2 0/1 - 1/1 * \log_2 1/1 = 0$
$Gain(SE, Flavor) = 1 - (3/6 * 0.91830) - (2/6 * 1) - (1/6 * 0) = 0.20752$
$Entropy(Color = Red) = -2/3 * \log_2 2/3 - 1/3 * \log_2 1/3 = 0.91830$
$Entropy(Color = Rose) = -1/3 * \log_2 1/3 - 2/3 * \log_2 2/3 = 0.91830$
$Gain(SE, Color) = 1 - (3/6 * 0.91830) - (3/6 * 0.91830) = 0.08170$

The subset that consists of examples whose flavor is equal to delicate is purely negative so we do not search for a test attribute any more. The subset having the value strong for flavor contains the examples identified as $E1$, $E3$ and $E6$ while examples having the value moderate consists of the exaples $E4$ and $E5$. Due to the fact that these subsets are not homogeneous, we look for a test attribute for each subset again.

As seen from Table 3.6 and Table 3.7, color attribute takes the highest information gain and deserves to be test attribute for both strong and moderate branches. The output tree is shown in the Figure 3.3.

Table 3.6. Entropies for wine attributes of the subset [E1, E3, E6]

Entropies & Information Gains
$Entropy(SE) = -2/3 * \log_2 2/3 - 1/3 * \log_2 1/3 = 0.91830$
$Entropy(Body = Medium) = -1/2 * \log_2 1/2 - 1/2 * \log_2 1/2 = 1$
$Entropy(Body = Full) = -1/1 * \log_2 1/1 - 0/1 \log_2 * 0/1 = 0$
$Gain(SE, Body) = 0.91830 - (2/3 * 1) - (1/3 * 0) = 0.25163$
$Entropy(Color = Red) = -2/2 * \log_2 2/2 - 0/2 * \log_2 0/2 = 0$
$Entropy(Color = Rose) = -0/1 * \log_2 0/1 - 1/1 * \log_2 1/1 = 0$
$Gain(SE, Color) = 0.91830 - -(2/3 * 0) - (1/3 * 0) = 0.91830$

Furthermore, several rules can be generated from the decision tree depicted in Figure 3.3. For instance, if flavor is strong and color is red, then the consumer has a tendency to accept the service, whereas if the flavor is strong but the color is rose then this service is possibly rejected. Following rules indicate that ID3 decision trees are successful to find disjunctive preferences in addition to conjunctives. Moreover, the rules are in a form of a disjunction of conjunctions. The rules for our example are:

- IF (((flavor = strong) and (color = red)) OR ((flavor = moderate) and (color = rose))) THEN *POSITIVE*
- IF (((flavor = strong) and (color = rose)) OR ((flavor = moderate) and (color = red) OR (flavor = delicate))) THEN *NEGATIVE*

The problem with this algorithm is that it is not an incremental algorithm, which means all the training examples should exist before learning. To overcome this problem, the system keeps consumer's requests throughout the negotiation interaction as positive examples and all counter-offers rejected by the consumer as negative examples. After each coming request, the decision tree is rebuilt. Without doubt, there is a drawback

Table 3.7. Entropies for wine attributes of the subset [E4, E5]

Entropies & Information Gains
$Entropy(SE) = -1/2 * \log_2 1/2 - 1/2 * \log_2 1/2 = 1$
$Entropy(Body = Light) = -1/2 * \log_2 1/2 - 1/2 * \log_2 1/2 = 1$
$Gain(SE, Body) = 1 - (2/2 * 1) = 0$
$Entropy(Color = Red) = -0/1 * \log_2 0/1 - 1/1 * \log_2 1/1 = 0$
$Entropy(Color = Rose) = -1/1 * \log_2 1/1 - 0/1 * \log_2 0/1 = 0$
$Gain(SE, Color) = 1 - (1/2 * 0) - (1/2 * 0) = 1$

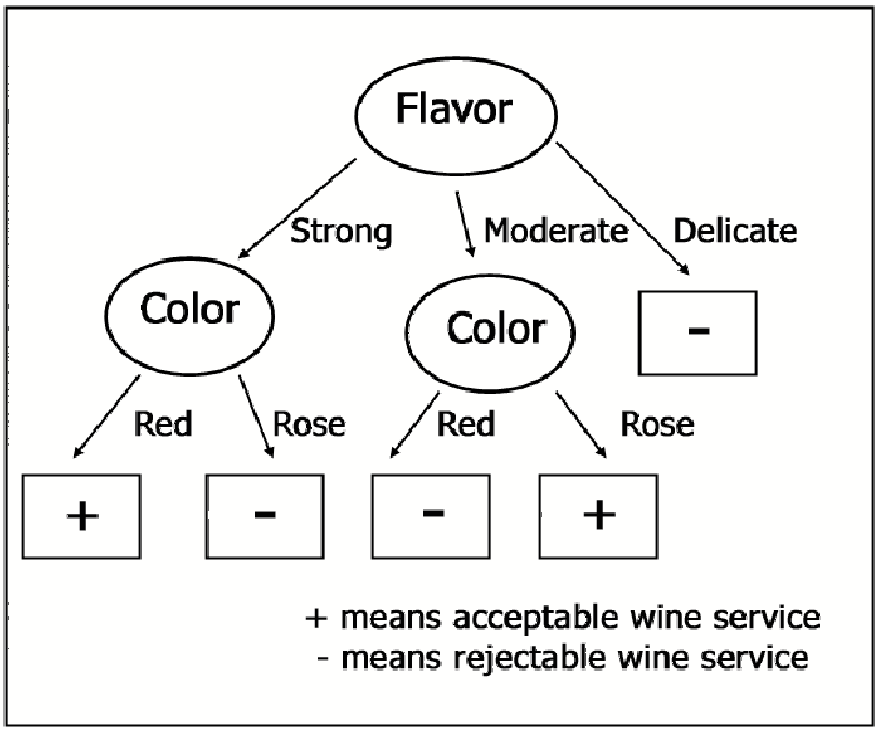

Figure 3.3. The decision tree for sample set

of reconstruction such as additional process load. Nevertheless, this process load is not significant for our purposes since ID3 Algorithm runs fast.

3.3. Similarity Estimation

Similarity can be estimated with a similarity metric that takes two entries and returns how similar they are. There are several similarity metrics used in case based reasoning system such as weighted sum of Euclidean distance, Hamming distance and so on [37].

Applied similarity metric affects the performance of the system while deciding which service is the closest to the consumer's request. We propose a new semantic similarity metric named *RP Similarity*. To see the performance gap, we present a variety of similarity measures. The similarity metrics that is investigated in this study are listed below:

- Tversky's similarity metric
- Resnik's semantic similarity metric
- Lin's semantic similarity metric
- Wu and Palmer's semantic similarity metric

3.3.1. Tversky's Similarity Metric

Firstly, we have used Tversky's similarity measure [38]. This metric compares two vectors in terms of the number of exactly matching features.

In Equation (3.3) [38], *common* represents the number of matched attributes whereas $different$ represents the number of the different attributes. Our current assumption is that α and β is equal to each other. As a future work, we will try

different weight values and select the best one by comparing them.

$$SM_{pq} = \frac{\alpha(common)}{\alpha(common) + \beta(different)} \tag{3.3}$$

To illustrate an example, we assume that the vector *(Full, Strong, Red)* represents the consumer's request and the vector *(Light, Strong, Rose)* represents any available service. The common feature between these vectors is only flavor with the value "Strong" and values of body and color are different in these vectors. As a result, the similarity is equal to 1/3. Here, when two features are compared, we assign zero for dissimilarity and one for similarity by omitting the semantic closeness among the feature values. However, we can use intermediate values to express the closeness of the attributes. For example, 0.6 can be used for the similarity between "Full" and "Medium" while 0.3 can be used for that between "Full" and "Light".

Assume that the second available service is *(Medium, Strong, Rose)*. The similarity between this vector and *(Full, Strong, Red)* is equal to 1/3 when the concrete values zero and one are used in estimation. While these two services seem to have the same similarity value, equally 1/3, they have different similarity when concerning the semantic closeness. "Medium" is semantically closer than "Light" when the desired wine body is "Full". In this case, the second available service *(Medium, Strong, Rose)* would be more desired to be an counter offer instead of the first, *(Light, Strong, Rose)*. In the following, the semantic similarity measures are discussed. But such extensions are not used in Tversky's original similarity metric.

3.3.2. Resnik's Semantic Similarity Metric

A taxonomy can be used while estimating semantic similarity between two concepts. Estimating semantic similarity in a Is-A taxonomy, can be done by calculating the distance between the nodes related to the compared concepts. The links among the nodes can be considered as distances. Then, the length of the path between the nodes indicates how closely similar the concepts are. An alternative estimation to use infor-

mation content in estimation of semantic similarity rather than edge counting method, was proposed by Philip Resnik [39]. By and large, Word-Net taxonomy is used in his work.

Formally, Resnik defines the semantic similarity between two concepts, c_1 and c_2 in Equation (3.4) where $S(c_1, c_2)$ is the set of concepts that subsumes both c_1 and c_2. According to the formula, the maximum value of $-\log P(c)$ where c is a concept in the set $S(c_1, c_2)$, is taken as the similarity between c_1 and c_2.

When the probability of being an instance of given concept increases, the informativeness decreases. The less general concept in taxonomy has the less probability value because it covers less instances. Thus, it would contain the most information. From this point, it can be said that the most specific concept that subsumes both c_1 and c_2 will give the higher informativeness than the superior concepts of this. Then the Equation (3.5) where c_0 is the most specific concept subsuming both c_1 and c_2, gives the same result with Equation (3.4).

$$Similarity(c_1, c_2) = \max_{c \in S(c_1, c_2)} [-\log P(c)] \tag{3.4}$$

$$Similarity(c_1, c_2) = -\log P(c_0) \tag{3.5}$$

For example, when we want to estimate the semantc similarity between "Red" and "Rose", the sample taxonomy depicted in Figure 3.4 can be used. Note that we describe this taxonomy in our shared ontology independently from other ontologies and Word-Net for being used in wine similarity. The most specific node that covers both "Red" and "Rose" is "ReddishColor". In our ontology, the word frequency is not defined. Instead, we take the proportion of concepts that the node owns including itself to all concepts. The probability for "ReddishColor" is equal to 3/6 since we have six concepts in all taxonomy and it covers "Red", "Rose" and "ReddishColor". As a result, the similarity between "Red" and "Rose" is equal to $-\log 3/6$, 1.0. The similarity estimated by this formula is not normalized between zero and one. For instance, the similarity between "Red" and "Red" is equal to $-\log 1/6$, 2.5850 whereas, that between

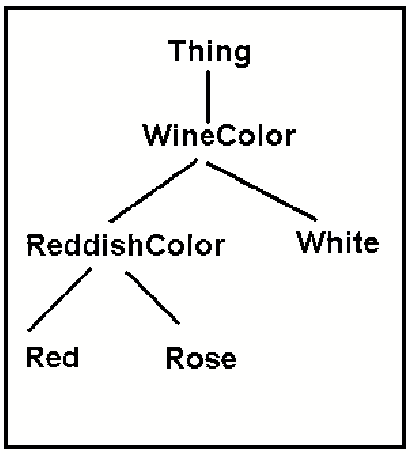

Figure 3.4. Sample wine color hierarchy

"Red" and "White" is equal to $-\log 5/6$, 0.2630. As is customary, we use logarithm 2 operation instead of logarithm 10.

3.3.3. Lin's Semantic Similarity Metric

Lin proposes a similarity metric close to Resnik's metric. The equation (3.6) [40] shows Lin's similarity where c_1 and c_2 are the compared concepts and c_0 is the most specific concept that subsumes both of them. Besides, $P(C)$ represents the probability of an arbitrary selected object belongs to concept C. Both Resnik's and Lin's similarity metric are based on the $\log P(c_0)$ but Lin divide this by the information content of the concepts. This operation means that Lin's metric is normalized and is allowed to take value between zero and one. On contrary, Resnik's similarity may take a value higher than one.

$$Similarity(c_1, c_2) = \frac{2 \times \log P(c_0)}{\log P(c_1) + \log P(c_2)} \tag{3.6}$$

With the same color taxonomy, the similarity between "Red" and "Rose" is equal to 0.3869 and the similarity between "Red" and "White" is estimated as 0.1018. Moreover, the similarity among same classes are calculated as 1. The results seem meaningful.

3.3.4. Wu and Palmer's Semantic Similarity Metric

Different from Resnik and Lin, Wu and Palmer use the distance between the nodes in IS-A taxonomy [41]. The semantic similarity is represented with Equation (3.7) [41]. Here, the similarity between c_1 and c_2 is estimated and c_0 is the most specific concept subsuming these classes. N_1 is the number of edges between c_1 and c_0. N_2 is the number of edges between c_2 and c_0. Moreover, N_0 is the number of IS-A links of c_0 from the root of the taxonomy.

$$Sim_{Wu\&Palmer}(c_1, c_2) = \frac{2 \times N_0}{N_1 + N_2 + 2 \times N_0} \tag{3.7}$$

For instance, when we estimate the similarity between *Red* and *Rose*, the c_0 would be *ReddishColor* and the distance of this node with the root is equal to 2 (N_0). N_1 is the distance of *ReddishColor* and *Red* and is 1. N_2 is the distance of *ReddishColor* and *Rose* ans is also 1. Then, the similarity is estimated as $((2 * 2)/(1 + 1 + 2 * 2))$, equally 0.6667. The similarity of *Red* and *White* is equal to 0.4. Similarly to Lin, the similarity of same conceps such as that between *Red* and *Red* is taken 1 by the formulla $((3 * 2)/(0 + 0 + 3 * 2))$.

3.3.5. RP Semantic Similarity Metric

Our approach to semantic similarity called *RP Similarity* is the estimation of relative distance in a taxonomy between two concepts. We have some intuitions about the similarity of nodes in taxonomies. These intuitions are listed below:

- Parent versus grandparents: Parent of a node is more similar to the node than grandparents of that. Generalization of a concept reasonably results in going further away that concept. The more general concepts are, the less similar they are. For example, *WineColor* is parent of *ReddishColor* and *ReddishColor* is parent of *Red*. Then, we expect the similarity between *ReddishColor* and *Red* to be higher than that of the similarity between *WineColor* and *Red*.

- Parent versus sibling: A node would have higher similarity to the parent of that than that to the sibling. For instance, *Red* and *Rose* are children of *ReddishColor*. In this case, we expect the similarity between *Red* and *ReddishColor* to be higher than that between *Red* and *Rose*.
- Sibling versus grandparents: The siblings are more similar when the relation between grandparents and grandchildren is compared. To illustrate, *WineColor* is grandparent of *Red*, and *Red* and *Rose* are siblings. Therefore, we possibly anticipate that *Red* and *Rose* are more similar than *WineColor* and *Red*.

As a taxonomy is represented in a tree, that tree can be traversed from the first concept being compared through the second concept. At starting node related to the first concept, the similarity value is constant and equal to one. This value is diminished by a constant at each node being visited over the path that will reach to the node including the second concept. The shorter path between the concepts exists, the higher similarity those have.

Relative distance is estimated in the following way. First of all, the similarity value at the first node is initialized with the value one. This value is multiplied by some constants whose value between zero and one. There are different constants for siblings and parents or children. Let m represent the constant for parents and n represent the constant for siblings. Except the sibling, m is multiplied by itself for each parent or children. Thus, the constant for grandparent of a node is equal to m^2. According to the above intuitions, our constants should be in a form $m > n > m^2$ where the value of m and n should be between zero and one. The estimation of similarity is explained in the Figure 3.5.

According to the algorithm, firstly the similarity is initialized with the value of one (line 1). If the concepts are equal to each other then, similarity will be 1 (lines 2-4). It is required to find the common parent of these two concepts and the distances to the common parent are estimated (lines 5-7). From these distances we substract 1 since we thought one edge is for sibling relation (lines 8-13). To clarify this, consider the relation between a person and her uncle. In this relation, there is one parent-child

Algorithm 4 RP Similarity Estimation Algorithm

Require: The constants should be $m > n > m^2$ where $m, n \in R[0, 1]$

1: $Similarity \leftarrow 1$
2: **if** $concept_1$ is equal to $concept_2$ **then**
3: Return $Similarity$
4: **end if**
5: $commonParent \leftarrow findCommonParent(concept_1, concept_2)$ { $commonParent$ is the most specific concept that covers both $concept_1$ and $concept_2$ }
6: $N1 \leftarrow findDistance(commonParent, concept_1)$
7: $N2 \leftarrow findDistance(commonParent, concept_2)$ {$N1$ and $N2$ are the number of links between the concept and parent concept}
8: **if** $N1 > 0$ **then**
9: $N1 \leftarrow N1 - 1$ {one of links is for the sibling so decrease the number of parent}
10: **end if**
11: **if** $N2 > 0$ **then**
12: $N2 \leftarrow N2 - 1$ {one of links is for the sibling so decrease the number of parent}
13: **end if**
14: **if** ($commonParent$ is equal to $concept1$) or ($commonParent$ is equal to $concept2$) **then**
15: $Similarity \leftarrow Similarity * m^{(N1+N2+1)}$
16: **else**
17: $Similarity \leftarrow Similarity * n * m^{(N1+N2)}$
18: **end if**
19: Return $Similarity$

Figure 3.5. RP Similarity Estimation Algorithm

relation between the person and her father. In addition, there is a sibling relation that exists between her father and her uncle. The common parent between the person and her uncle is her grandfather and there are two edges between the person and her grandfather. We emphasize that one is for the sibling relation (father-uncle).

If one of the concept is equal to common parent, there is no sibling relation between to concept. Therefore, the constant for the sibling is not included in the similarity estimation. For each level, we multiply the similarity by m. As a result, we decrease the similarity at each level with the rate of m (lines 14-15). Otherwise, there should be one sibling relation between the nodes and we consider distance of parent-child in estimation (line 17).

To illustrate this similarity, the sample taxonomy depicted in Figure 3.6 is used. Some similarity estimations belonging to this taxonomy are given in Table 3.8. In this example, m is accepted as $2/3$ and n is taken as $4/7$.

Table 3.8. Sample similarity estimation over sample taxonomy

$Similarity(E, G) = 1 * (2/3) = 0.6666667$
$Similarity(F, G) = 1 * (4/7) = 0.5714286$
$Similarity(B, G) = 1 * (2/3)^2 = 0.44444445$
$Similarity(A, G) = 1 * (2/3)^3 = 0.2962963$
$Similarity(D, G) = 1 * (4/7) * (2/3) = 0.3809524$
$Similarity(C, G) = 1 * (4/7) * (2/3)^2 = 0.25396827$
$Similarity(H, G) = 1 * (4/7) * (2/3)^3 = 0.16931216$
$Similarity(I, G) = 1 * (4/7) * (2/3)^3 = 0.16931216$

For sample color ontology, RP Similarity finds the similarity between *Red* and *Rose* as 0.5714 and the similarity between *Red* and *White* is estimated as 0.3810. Againg the similarity between the same concepts is equal to one.

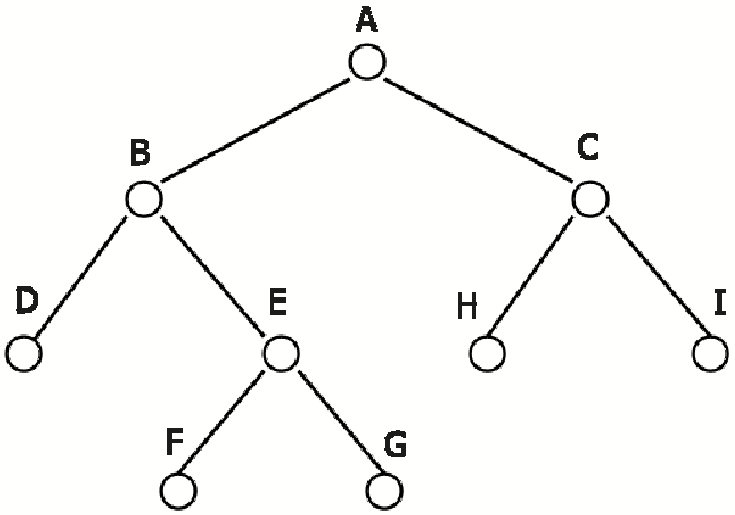

Figure 3.6. Sample taxonomy for similarity estimation

For all semantic similarity metrics in our architecture, the taxonomy for features is held in the shared ontology. In order to evaluate the similarity of feature vector, we firstly estimate the similarity for feature one by one and take the average sum of these similarities. Then the result is equal to the semantic similarity of all feature vector.

3.4. Offering Service

Making the right service offers is the most important process in content-oriented negotiation. To generate the best offer, the producer agent uses its service ontology and the inductive learning. To accomplish it, we propose to use Modified Candidate Elimination Algorithm or to use ID3 Decision Tree. Alternative offering mechanisms are also investigated in order to see which system works better.

All offering service algorithms used here are listed in following order:

- Random Offering Service: It offers randomly from the available service list.
- Offering Service considering only the current request: It decides according to the similarity to the current request so it is called offering service using SCR.

- Offering Service using Version Space (VS) : Candidate Elimination Algorithm is used here.
- Offering Service using Modified Version Space (MVS): The modified version of Candidate Elimination Algorithm is used.
- Offering Service using Decision Trees (DT): ID3 Algorithm is used.

We use a variety of similarity metrics separately for testing purposes. Note that in this section, we use the Tversky's metric in our example.

3.4.1. Random Offering

Another alternative offering is to offer any service from the service list. In this system, the counter offer is randomly selected. Therefore, the number of the services in the available service list affects the performance of the system. If there is a huge number of services meeting the consumer's expectation, the system works well. On the other hand, in the case of having a few convenient services in the inventory, the performance of the system may decrease drastically.

3.4.2. Offering Service via Similarity to Current Request

With the purpose of comparision, we also investigate a system, which offers the most similar service to the current request of the consumer. In this system, we only consider the current request and ignore all past requests.

3.4.3. Offering Service via Version Space Based Learning

To generate the best offer, the producer agent uses its service ontology and an inductive learning. The offering service algorithm is the same for both the original Candidate Elimination Algorithm and the Modified Candidate Elimination Algorithm.

When producer receives a request from the consumer, the learning set of the producer is trained with this request as a positive sample. The learning components,

the most specific set S and the most general set G are actively used in offering service. The most general set, G is used by the producer in order to avoid offering the services, which will be rejected by the consumer agent. In other words, it filters the service set from the undesired services, since G contains hypotheses that are consistent with the requests of the consumer. The most specific set, S is used in order to find best offer, which is similar to the consumer's preferences. Since the most specific set S holds the previous requests and the current request, estimating similarity between this set and every service in the service list is very convenient to find the best offer from the service list. This algorithm used in offering a service to the consumer is given in Figure 3.7.

When the consumer starts the interaction with the producer agent, producer agent loads all related services to the service list object. This list constitutes the provider's inventory of services. Upon receiving a request, if the producer can offer an exactly matching service, then it does so. For example, for a wine this corresponds to selling a wine that matches the specified features of the consumer's request identically. When the producer cannot offer the service as requested, it tries to find the service that is most similar to the services that have been requested by the consumer during the negotiation. To do this, the producer has to compute the similarity between the services it can offer and the services that have been requested.

The system using the original Version Space is called Similarity with Version Space (SVS) whereas the system using Modified Version Space is named Similarity with Modified Version Space (SMVS).

The similarity metrics that we have discussed in Section 3.3 are used for comparing two feature vectors. In our system while the list of services that can be offered by the producer are each a feature vector, the most specific set S is not a feature vector. S consists of hypotheses of feature vectors. Therefore, we estimate the similarity of each hypothesis inside the most specific set S and then take the average of the similarities. Moreover, each hypothesis in S may involve a different number of service instances.

Algorithm 5 Offering Service Algorithm via Version Space Based Learning

Require: The learning algorithm is trained with the request as a positive example

```
1: if it is the first time then
2:    ServiceList ←All related available services from service ontology {the producer
      agent can provide }
3: end if
4: if any service in ServiceList is equal to the request then
5:    Offer service to the consumer
6: else
7:    Remove each service in ServiceList which is not covered by the general sets in
      learning object
8:    for all service in ServiceList do
9:       Estimate similarity between service and the most specific set in learning object
10:      Hold the maximum similarity in MaxSim
11:      Hold the count of services owing the maximum similarity in ServiceCount
12:   end for
13:   if ServiceCount > 1 then
14:      Find rating values of s whose similarity is equal to MaxSim
15:      Offer the services whose rate value is the biggest
16:   else
17:      Offer the service whose similarity is equal to MaxSim
18:   end if
19: end if
```

Figure 3.7. Offering Service Algorithm

Assume that S contains the following two hypothesis: { {*Light, Moderate, (Red, White)*} , {*Full, Strong, Rose*}}. Take service s as *(Light, Strong, Rose)*. Then the similarity of the first one is equal to 1/3 and the second one is equal to 2/3 in accordance with Equation (3.3). Normally, we take the average of it and obtain $(1/3 + 2/3)/2$, equally 1/2. However, the first hypothesis involves the effect of two requests and the second hypothesis involves only one request. As a result, we expect the effect of the first hypothesis to be greater than that of the second.

Therefore, we calculate the average similarity by considering the number of samples that hypotheses cover. Let c_h denote the number of samples that hypothesis h covers and $(SM_{(h,service)})$ denote the similarity of hypothesis h with the given service. We compute the similarity of each hypothesis with the given service and weight them with the number of samples they cover. We find the similarity by dividing the weighted sum of the similarities of all hypotheses in S with the service by the number of all samples that are covered in S.

$$AVG - SM_{(service,S)} = \frac{\sum_{|h|}^{|S|} \left(c_h * SM_{(h,service)}\right)}{\sum_{|h|}^{|S|} c_h} \tag{3.8}$$

For the above example, the similarity of *(Light, Strong, Rose)* with the specific set is $(2 * 1/3 + 2/3)/3$, equally 4/9. The possible number of samples that a hypothesis covers can be estimated with multiplying cardinalities of each attribute. For example, the cardinality of the first attribute is two and the others is equal to one for the given hypothesis such as {*Light, Moderate, (Red, White)*}. When we multiply them, we obtain two $(2 * 1 * 1 = 2)$.

Offering a service is actually similar to case-based approach which is nearly done in Section 3.4.2. We select the feature values of the service vector in the service list obtained from the ontology and filtered by the most general set G, which is the most similar to the specific set in our learning algorithm whereas in case-based approach, the most similar one to the current the request is chosen by losing the past request

information. In other words, when we investigate the similarity, we do not only consider the current request, but also look at the previous request of the consumer by the help of the most specific set covering all requests during the negotiation phase.

If there is more than one service with the maximum similarity, the producer agent chooses one of them in accordance with the rating value of it; the higher rating is more attractive for the consumer agent. After receiving a service offer, there are two options for the consumer: to accept or reject. If the consumer rejects, the producer trains the learning set with the offer as a negative sample. This will continue until the consumer takes a service or until a certain time interval.

3.4.4. Offering Service via Decision Tree

As an alternative to Version Space based learning, decision trees can be applied in our architecture for learning consumer's preferences. Here, ID3 Decision Tree Algorithm is used for classification of services such as positive and negative. At each time of coming request, the decision tree is rebuilt by the system.

The offering service mechanism is similar to that offered in Section 3.4.3 with two changes. One of the changes in Offering Service Algorithm is that we remove the services from the available service list, which is classified as negative according to the decision tree instead of removing those that G does not cover (line 7). The second change in the algorithm is not to estimate the similarity to S (line 9). Instead, we keep all positive service descriptions during the interaction. Then, we estimate the similarity to each other and take the average sum of them. There is no more change in the algorithm.

To illustrate this, assume we have the training examples shown in Table 3.4. Then the consumer requests *(Light, Strong, Red)*. The system rebuilds the tree and luckily the tree is same as depicted one in Figure 3.3. Our available service list would not include such services *(Light, Strong, Rose)* since that service would be defined as negative by the decision tree and that would be removed from the list.

4. DEVELOPED SYSTEM

We have implemented our architecture in Java. To ease testing of the system, the consumer agent has a user interface that allows us to enter various requests. The producer agent is fully automated and the learning and service offering operations work as explained before. In this section, we explain the implementation details of the developed system.

4.1. Shared Ontology

The shared ontology is the modified version of the *Wine Ontology* [36]. It includes the description of wine as a concept and different types of wine. All participants of the negotiation use this ontology for understanding each other. It is used as a medium for communication.

The description of the wine concept in OWL is shown in Figure 4.1. According to this description, seven properties make up the wine concept. Consumer agent and producer agent obtain the range information about these properties by querying the ontology. Thus, all possible values for the components of the wine concept such as color, body, sugar and so on can be reached by both agents. Also a variety of wine types are described in this ontology such as *Burgundy*, *Chardonnay*, *CheninBlanc* and so on. Intuitively, any wine type described in the ontology also represents a wine concept. This allows us to consider instances of *Chardonnay* wine as instances of *Wine* class.

In addition to wine description, the hierarchical information of some features can be deduced from the ontology. For instance, we can represent the information *Europe Continent* covers *Western Country*. *Western Country* covers *French Region*, which covers some territories such as *Loire*, *Bordeaux* and so on. This hiearchical information will be used in estimation of semantic similarity. In this part, some reasoning can be made such as if a concept X covers Y and Y covers Z, then concept X covers Z.

```
<owl:Class rdf:ID="Wine">
    ...
  <rdfs:subClassOf>   <owl:Restriction>
     <owl:onProperty rdf:resource="#hasMaker" />
     <owl:allValuesFrom rdf:resource="#Winery" />
   </owl:Restriction>  </rdfs:subClassOf>
  <rdfs:subClassOf> ... <owl:onProperty rdf:resource="#madeFromGrape" /> ...
  <rdfs:subClassOf> ... <owl:onProperty rdf:resource="#hasSugar" />     ...
  <rdfs:subClassOf> ... <owl:onProperty rdf:resource="#hasFlavor" /> ...
  <rdfs:subClassOf> ... <owl:onProperty rdf:resource="#hasBody" /> ...
  <rdfs:subClassOf> ... <owl:onProperty rdf:resource="#hasColor" /> ...
  <rdfs:subClassOf>  <owl:Restriction>
     <owl:onProperty rdf:resource="#locatedIn"/>
     <owl:someValuesFrom rdf:resource="#Region"/>
  </owl:Restriction>  </rdfs:subClassOf>
  </owl:Class>
  ...
```

Figure 4.1. Wine description in OWL

For example, *Europe Continent* covers *Bordeaux*. Moreover, we also represent n-ary relations of some concepts in modified *Wine Ontology* with the purpose of keeping similarity relationship among some features. For example, for the values of body feature, we hold the similarity in the ontology.

For some features such as body, flavor and sugar, there is no hierarchical information. Thus, we give the reasonable similarity values for these features and represent them in a n-ary form. Except these features, we create a tree representing the hiearchy among the values of features by using the information kept in the ontology. In the following parts, we will investigate this matter in detail.

4.1.1. Semantic Similarity for Body

The possible values declared in the ontology for body feature are *Light*, *Medium* and *Full*. In the developed system, we give the similarities by concerning the levels of body. There are three levels and the distance between two levels approximately corresponds to 0.3. These similarities are listed in Table 4.1.

Table 4.1. Semantic similarities for body

Body 1	Body 2	Similarity
Light	Full	0.3
Medium	Light	0.6
Full	Medium	0.6

The question is how we can represent these similarities with OWL. With RDF or OWL, we can represent axioms in triples such as (subject, predicate, object). For instance, we can write a triple such as (Wine, hasBody, Light) where *Wine* is the subject, *hasBody* is the predicate and *Light* is the object. Here, *hasBody* is declared as a functional object property in OWL. OWL properties allow us to relate one individual to another individual or a value. Thus, we cannot declare an OWL property, which relates two individuals and a value.

However, we need to add information such as that the individual *Light* is similar to *Full* with the value of 0.3. We cannot represent this information with a triple since we have three two individuals, one value and a relation. In detail, a binary relation exists between the individuals, *Light* and *Full*, and 0.3 is an additional information about this relation. Thus, we need to represent a n-ary relation here.

One way to accomplish this is to use an individual of the relation, which relates the things and an actual relation is the class of these individuals [42]. In detail, we take the main object in one way and we create an instance, which describes the relation itself. Then, we relate the main object to this relation object.

```
        ... <owl:Class rdf:ID="WineBodyRelation">
<rdfs:subClassOf>
<owl:Restriction>  <owl:someValuesFrom rdf:resource="#WineBody"/>
<owl:onProperty> <owl:FunctionalProperty rdf:about="#similarWithWineBody"/>
</owl:onProperty> </owl:Restriction> </rdfs:subClassOf>
<rdfs:subClassOf> <owl:Restriction>  <owl:onProperty>
 <owl:FunctionalProperty rdf:about="#similarityValue"/>   </owl:onProperty>
 <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction>  </rdfs:subClassOf>   </owl:Class>
<owl:FunctionalProperty rdf:ID="similarWithWineBody">
  <rdfs:range rdf:resource="#WineBody"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>  ...
  <WineBody rdf:ID="Light">
   <hasWineBodyRelation>
        <WineBodyRelation rdf:nodeID="WineBodyRelation_3">
          <similarWithWineBody rdf:resource="#Full" />
          <similarityValue  rdf:datatype="&xsd;float">0.3</similarityValue>
         </WineBodyRelation>
    </hasWineBodyRelation>
   </WineBody> ...
```

Figure 4.2. Wine body relation in OWL

For the above example, *Light* has similarity to *Full* with the value of 0.3. We create a relation class such as *WineBodyRelation* that has two functional property: *similarWithWineBody* and *similarityValue*. An instance of this class holds *Full* and 0.3. The class *WineBody* has a property called *hasWineBodyRelation* whose range is a type of *WineBodyRelation*. Figure 4.2 shows the OWL code of this part. The relation is depicted in Figure 4.3.

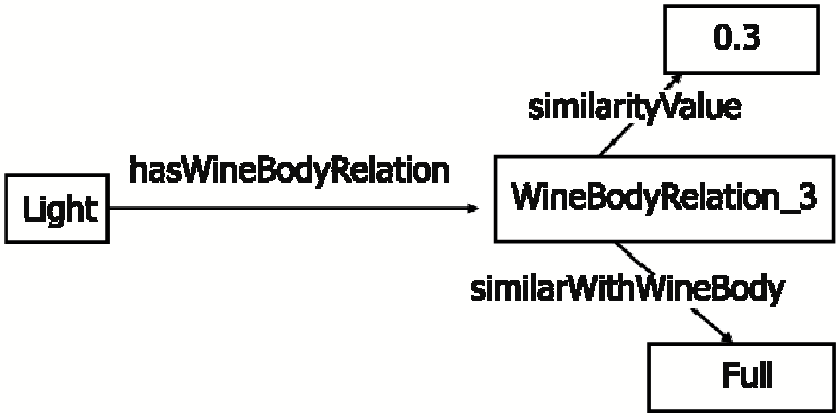

Figure 4.3. Wine body relation

4.1.2. Semantic Similarity for Sugar

Sweetness degree of wine can be categorized in five levels: dry, offdry, slightly sweet, medium sweet and sweet [43]. When the similarity value one is divided into five equal parts, the distance between each level would be equal to 0.2. Thus, we accept that the similarity between two categories decreases by 0.2 at each level, i.e., equal distance among levels. However, the original wine ontology does not contain slightly sweet and medium sweet. In our ontology, we only use *Dry*, *OffDry* and *Sweet*. We give the similarities by concerning all levels of sugar discussed above. Table 4.2 indicates the similarities.

Table 4.2. Semantic similarities for sugar

Sugar 1	Sugar 2	Similarity
Sweet	Dry	0.2
OffDry	Sweet	0.4
Dry	OffDry	0.8

4.1.3. Semantic Similarity for Flavor

The valid values for flavor are *Delicate*, *Moderate* and *Strong*. We give the similarities by concerning the levels of flavor similarly to that is done for body and sugar. The similarities for flavor are given in Table 4.3.

Table 4.3. Semantic similarities for flavor

Flavor 1	Flavor 2	Similarity
Delicate	Strong	0.3
Moderate	Delicate	0.6
Strong	Moderate	0.6

4.1.4. Semantic Similarity for Color

We construct a taxonomy for the color attribute from the descriptions in the ontology. The taxonomy is depicted in Figure 4.4. Then we apply semantic similarity metrics discussed in Chapter 3. The similarities are given in Table 4.4.

Table 4.4. Semantic similarities for color

Color 1	Color 2	By Wu Palmer	By Resnik	By Lin	By RP
Red	Red	1.0	2.5849626	1.0	1.0
Rose	Rose	1.0	2.5849626	1.0	1.0
White	White	1.0	2.5849626	1.0	1.0
Red	White	0.4	0.2630344	0.1017556	0.3809524
Red	Rose	0.6666667	1.0	0.3868528	0.5714286
Rose	White	0.4	0.2630344	0.1017556	0.3809524

4.1.5. Semantic Similarity for Winery

We categorize wineries as expensive, moderate and cheap by considering the price. Then we create the hierarchy shown in Figure 4.5. Note that the distribution

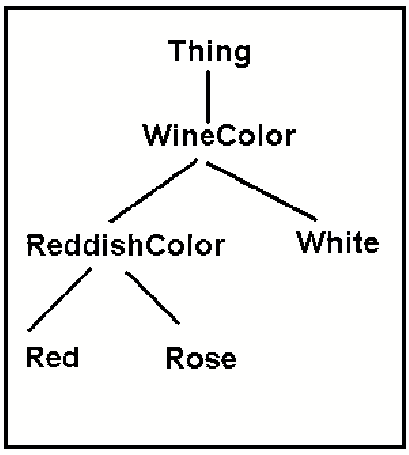

Figure 4.4. Wine color hierarchy

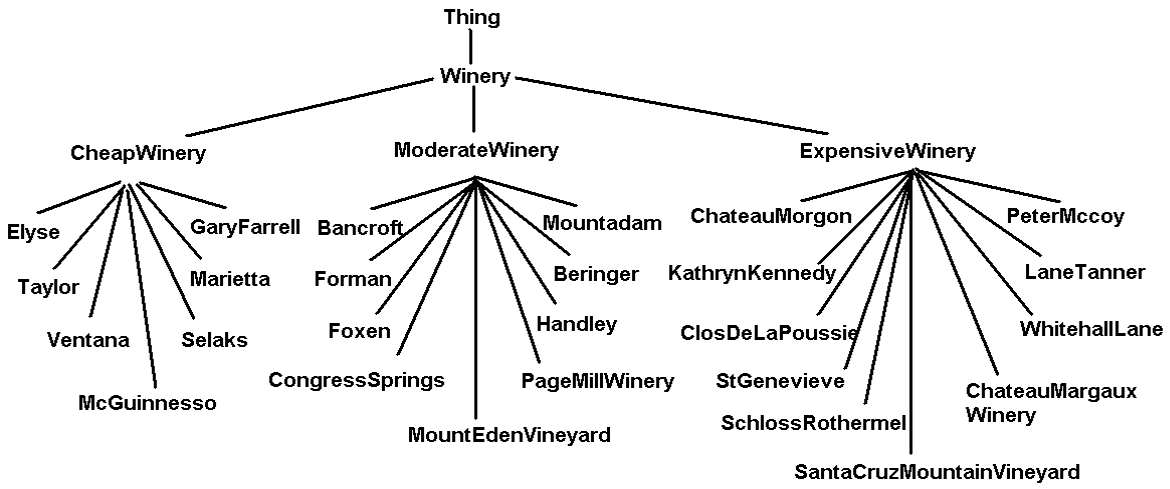

Figure 4.5. Winery hierarchy

of wineries are performed arbitrarily. For instance, we do not know about the price of any Bancroft's product in real life but it will be taken as moderate in our system as an assumption. Some semantic similarities estimated for winery are indicated in Table 4.5.

4.1.6. Semantic Similarity for Grape

There are two types of wine grape: white and red. According to the information in Oxford University Wine Society Home Page [44], we build the taxonomy depicted in Figure 4.6. Some estimated semantic similarities for grapes are listed in Table 4.6.

Table 4.5. Semantic similarities for winery

Winery 1	Winery 2	By Wu Palmer	By Resnik	By Lin	By RP
Taylor	Taylor	1.0	4.95420	1.0	1.0
Taylor	Selaks	0.66667	1.95420	0.39445	0.57143
Taylor	Handley	0.33334	0.04731	0.00955	0.25397
Taylor	PeterMccoy	0.33334	0.04731	0.00955	0.25397
Foxen	Foxen	1.0	4.95420	1.0	1.0
Foxen	ChateauMorgon	0.33334	0.04731	0.00955	0.25397
Foxen	Mountadam	0.66667	1.63227	0.32947	0.57143
Foxen	Ventana	0.33334	0.04731	0.00955	0.25397

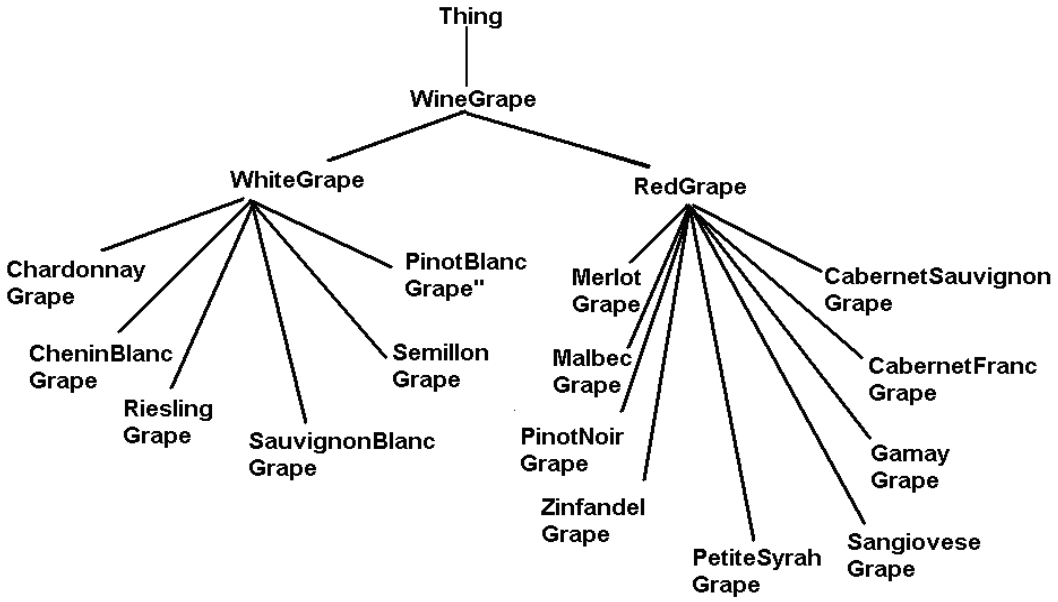

Figure 4.6. Wine grape hierarchy

Table 4.6. Semantic similarities for grape

Grape 1	Grape 2	Wu Palmer	Resnik	Lin	RP
ChardonnayGrape	ChardonnayGrape	1.0	4.3219	1.0	1.0
ChardonnayGrape	CheninBlancGrape	0.6667	1.5146	0.3504	0.5714
ChardonnayGrape	ZinfandelGrape	0.3334	0.0740	0.0171	0.2540
SemillonGrape	SemillonGrape	1.0	4.3219	1.0	1.0
SemillonGrape	PinotNoirGrape	0.3334	0.0740	0.0171	0.2540
SemillonGrape	RieslingGrape	0.6667	1.5146	0.3504	0.5714

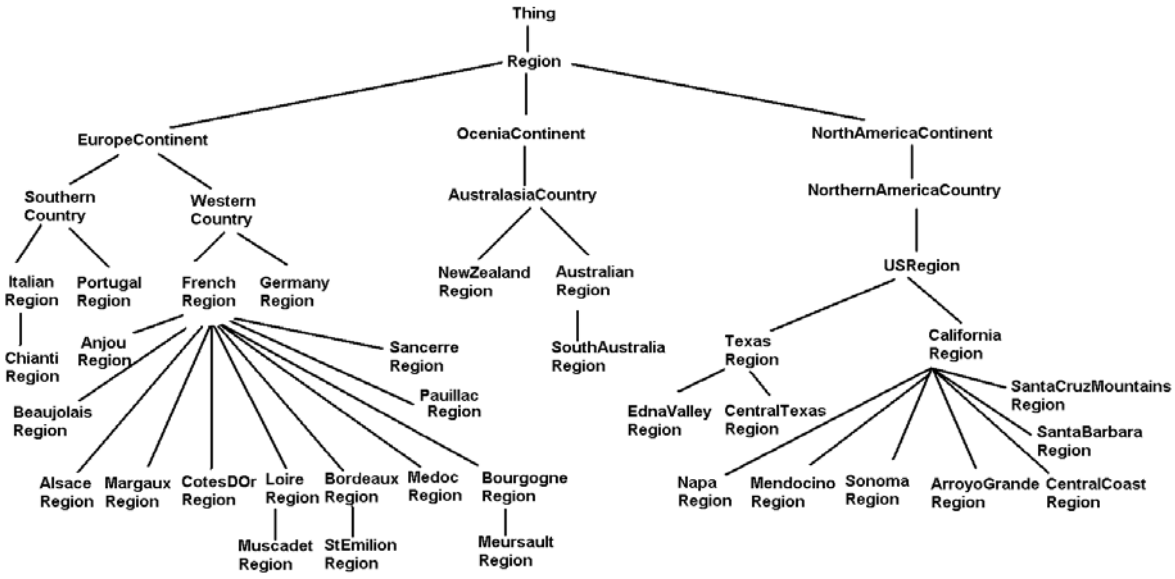

Figure 4.7. Wine region hierarchy

4.1.7. Semantic Similarity for Region

We also describe a region hierarchy being used in similarity estimation. Figure 4.8 involves some part of OWL code, which describes related parts to region. We use the element *subclass* to describe some part of the hieararchy and declare a transitive property "covers", which helps construct the hierarchy. If a region or a territory covers $XRegion$ territory, in the hierarchy $XRegion$ will be a child of that. Table 4.7 shows some similarities. Figure 4.7 shows the taxonomy for region.

Table 4.7. Semantic similarities for region

Region 1	Region 2	Wu Palmer	Resnik	Lin	RP
ChiantiRegion	ChiantiRegion	1.0	5.4263	1.0	1.0
ChiantiRegion	ItalianRegion	0.8889	4.4263	0.8985	0.6667
ChiantiRegion	PortugalRegion	0.6667	3.4263	0.6314	0.3810
ChiantiRegion	FrenchRegion	0.4445	0.9668	0.2784	0.1693
ChiantiRegion	LoireRegion	0.4	0.9668	0.1963	0.1129
ChiantiRegion	MuscadetRegion	0.3636	0.9668	0.1782	0.0752
ChiantiRegion	USRegion	0.2222	0.0339	0.0093	0.0752
ChiantiRegion	TexasRegion	0.2	0.0339	0.0073	0.0502
ChiantiRegion	EdnaValleyRegion	0.1818	0.0339	0.0063	0.0334
StEmilionRegion	StEmilionRegion	1.0	5.4263	1.0	1.0
StEmilionRegion	BordeauxRegion	0.9091	4.4263	0.8985	0.6667
StEmilionRegion	PauillacRegion	0.7273	1.5194	0.2800	0.3810
StEmilionRegion	FrenchRegion	0.8	1.5194	0.4375	0.4445
StEmilionRegion	GermanyRegion	0.6	1.3388	0.2467	0.2540
StEmilionRegion	AustralianRegion	0.2	0.0339	0.0069	0.0502
StEmilionRegion	SouthAustraliaRegion	0.1818	0.0339	0.0063	0.0334
StEmilionRegion	SantaBarbaraRegion	0.1667	0.0339	0.0063	0.02230

```
<owl:Class rdf:ID="WesternCountry">
  <rdfs:subClassOf rdf:resource="#EuropeContinent"/>
</owl:Class> ....
<owl:Class rdf:about="#EuropeContinent">
  <rdfs:subClassOf rdf:resource="#Region"/>
</owl:Class> ...
<owl:Class rdf:ID="Territory"/>
<owl:ObjectProperty rdf:ID="covers">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  <rdfs:range rdf:resource="#Territory" />
</owl:ObjectProperty> ...
<WesternCountry rdf:ID="FrenchRegion">
   <covers rdf:resource="#BeaujolaisRegion" /> ....
</WesternCountry>  ....
```

Figure 4.8. Region description in OWL

4.2. Data Repository Ontology

In shared wine ontology, we describe the wine classes and their properties. Also, that ontology contains a variety of wine instances. One of them is shown in Figure 4.9. Moreover, producer agent needs another ontology as a data repository to keep its available services. This specific ontology is called *WineStock Ontology* in our system and extends the *Wine Ontology*.

WineStock Ontology describes a product class as *WineProduct*. This class is necessary for the producer to record the wines that it sells; i.e., its inventory information. Ontology involves the individuals of this class. The individuals represent available services that the producer owns. Figure 4.10 shows the description of *WineProduct* and a sample individual of that class.

```
<Beaujolais rdf:ID="ChateauMorgonBeaujolais">
   <hasMaker  rdf:resource="#ChateauMorgon" />
   <madeFromGrape rdf:resource="#GamayGrape" />
   <hasSugar  rdf:resource="#Dry" />
   <hasFlavor rdf:resource="#Delicate" />
   <hasBody   rdf:resource="#Light" />
   <hasColor  rdf:resource="#Red" />
   <locatedIn rdf:resource="#BeaujolaisRegion" />
 </Beaujolais>
```

Figure 4.9. A sample wine instance in OWL

The count value keeps track of the number of products in inventory and is used to check availability of the product. The rating value is an indicator of the quality of the wine product. Among two similar wines, the one with higher rating value is preferred over the other. The price of the wine is deliberately left out to emphasize that we are concerned with the value of the service.

We prepare two separate *WineStock* ontologies for testing. In our first ontology, there are 19 available wine products. Then we increase the number of products up to 50 in the second ontology. Table B shows the services in the first dataset and Table B indicates the additional services to the first data set that exist only in the second data set. In both tables, the symbol "R" shows the rating value for that product. The value of ratings is between zero and five.

4.2.1. Consumer Agent

The consumer agent presents an interface to the customer, which is constructed dynamically from the ontology. By using the class URI of the wine, the components that makes up the wine service and their possible values can be obtained. Subclass properties and some simple inferences are used here. Figure 4.11 presents the consumer agent's interface.

```
<owl:Class rdf:ID="WineProduct"/>
<owl:DatatypeProperty rdf:ID="Count">
 <rdfs:domain rdf:resource="#WineProduct"/>
 <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Rating">
  <rdfs:domain rdf:resource="#WineProduct"/>
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="hasWine">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#WineProduct"/>
  <rdfs:range rdf:resource="&vin;Wine"/>
</owl:ObjectProperty>
<WineProduct rdf:ID="WineProduct1">
  <Count rdf:datatype="&xsd;nonNegativeInteger">1</Count>
  <Rating rdf:datatype="&xsd;nonNegativeInteger">4</Rating>
  <hasWine rdf:resource= "&vin;ChateauMorgonBeaujolais"/>
</WineProduct> ...
```

Figure 4.10. WineProduct description and a sample instance in OWL

4.2.2. Producer Agent

In addition to obtaining all components of the wine service, producer agent uses ontology in order to build hierarchical trees of some features such as region, color, winery and grape, which are being used in semantic similarity estimation. Some simple inferences are performed by using *subClassOf* properties and transitive properties such as *covers*. Jena's inference engine can directly perform *subClassOf* inference. For example, *WineColor* is a class and *ReddishColor* is subclass of it. When we query the individuals of *WineColor*, in addition to instances of *WineColor* such as *White*, the individuals of *ReddishColor* such as *Red* and *Rose* are retrieved by Jena's inference engine.

Figure 4.11. Consumer interface

Moreover, we can use SPARQL query language [25] with JENA. Figure 4.12 shows a JAVA code including SPARQL query running with JENA. Here, we try to query the similarities of a property such as body, flavor and sugar. In the query, $firstTerm$ and $secondTerm$ will be an instance of a property. For instance, *Sweet*, *OffDry* and *Dry* are possible instances for sugar property. The output of this query will return a pair of properties with their similarity values. For instance, (*Light*, *Full*, 0.3) may be one of the items returned by the query.

The available services of the producer agent are loaded at the beginning of the program by using the *WineStock Ontology*. In addition to the existing similarities in ontology, the provider keeps the hierarchy of some featues and uses it in semantic estimation. Producer agent tries to make the best counter offer, which is similar to the request in an harmony with consumer's preferences.

```
String queryString =
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> "+
"SELECT ?firstTerm ?secondTerm ?similarity " +
"WHERE {" + "   ?firstTerm <"+semanticAddress+"#has"+
   propertyPair.propertyName+"Relation> ?x . " + " ?x <"+semanticAddress+
   "#similarWith"+propertyPair.propertyName+"> ?secondTerm . " +
   " ?x <"+semanticAddress+"#similarityValue> ?similarity . " + " }";

Query query = QueryFactory.create(queryString);
QueryExecution qe = QueryExecutionFactory.create(query,this.ontologyModel);
ResultSet results = qe.execSelect();
```

Figure 4.12. Running a SPARQL query in JENA

4.3. Some Illustrations

We give several illustrations here to clarify the negotiation mechanism. The given examples use the first data set involving 19 available products. Tversky's similarity measure is used in first two examples. Firstly, we give an example of the system using Modified Version Space, MVS. Secondly, the example will be investigated for the system but this system uses ID3 Algorithm instead of MVS. Thirdly, the same example is examined by the MVS system but similarity metric used in this example is RP Similarity.

4.3.1. Example of MVS system

Consider a customer that is planning on buying wine. For the customer, the color, location and winery of the wine are more important than the remaining features such as the body, flavor and so on. In other words, the consumer can be convinced to buy a wine product whose flavor is different from the one that the consumer initially specifies in its request, but she cannot be convinced to buy the wine whose color is different from her initial specification. Obviously, these preferences are not known by

the service provider ahead of time.

Table 4.8 depicts the interactions between the consumer and the producer. The consumer starts the negotiation by generating the following request (Elyse, Merlot-Grape, OffDry, Strong, Medium, Red, NapaRegion) where Elyse is the name of winery, MerlotGrape is the type of grape that wine is made from, OffDry indicates the sugar level, Strong is the flavor of wine where Medium is an indicator for the body of the wine, the color of wine is Red and NapaRegion is the region of the wine.

When this request is taken by the producer, it firstly trains its learning set, treating this request as a positive sample as explained before. Since this is the only positive example so far, it constitutes the most specific set S in accordance with the modified CEA. Then, it estimates the similarities of each available service in the service list of the producer shown in Table B with the most specific set. The output of the similarities estimated by the producer agent using Equation (3.3) is displayed in Table 4.9.

Table 4.8. An example negotiation

R-1	[Elyse,MerlotGrape,OffDry,Strong,Medium,Red,NapaRegion]
O-1	[Forman,CabernetSauvignonGrape,Dry,Strong,Medium,Red,NapaRegion]
R-2	[Elyse,MalbecGrape,OffDry,Strong,Medium,Red,NapaRegion]
O-2	[WhitehallLane,CabernetFrancGrape,Dry,Moderate,Medium,Red,NapaRegion]
R-3	[Elyse,MalbecGrape,OffDry,Strong,Light,Red,NapaRegion]
O-3	[Elyse,ZinfandelGrape,Dry,Moderate,Full,Red,NapaRegion]

According to the similarities of the first time, the most similar product is selected as $P3$ having the maximum similarity value 0.571 ($= 4/7 = (common/(common + different))$). This service supplies the same color and location information with the initial request. However, its winery is different from the one that the consumer agent strictly wants. Therefore, the consumer rejects the offer and makes another request, which is also in accordance with the customer's preferences (i.e., only the grape type is

Table 4.9. Estimated similarities for MVS example

Product ID	After Req.1	After Req.2	After Req.3
P1	0.143	0.143	0.190
P2	0.429	**0.429**	-
P3	**0.571**	-	-
P4	0.286	0.286	0.238
P5	0.429	**0.429**	0.380
P6	0.286	0.286	0.286
P7	0.286	0.286	0.238
P8	0.143	0.143	0.143
P9	0.143	0.143	0.143
P10	0.143	0.143	0.095
P11	0.143	0.143	0.095
P12	0.0	0.0	0.0
P13	0.286	0.286	0.238
P14	0.143	0.286	0.333
P15	0.143	0.286	0.238
P16	0.143	0.143	0.143
P17	0.0	0.0	0.048
P18	0.429	**0.429**	**0.429**
P19	0.286	0.286	0.286
Max. Similarity:	**0.571**	**0.429**	**0.429**

changed). At this point, the producer inserts the rejected offer as a negative example into its learning algorithm and removes it from the service list. Following this, the producer recalculates the similarity of the services in its service list with S, which is equivalent to { *(Elyse , [MerlotGrape, MalbecGrape], OffDry, Strong, Medium, Red, NapaRegion)* }. At this time, there are three products with the maximum similarity value 0.429. Maximum estimated similarities are:

- $Similarity of P2 = (2 * 3/7)/2 = 0.429$
- $Similarity of P5 = (2 * 3/7)/2 = 0.429$
- $Similarity of P18 = (2 * 3/7)/2 = 0.429$

The producer prefers to offer the second product since its rating value is higher than the other products. However, the winery requirement of the customer is not supported by this offer. Therefore, the consumer agent will reject it. The next request, the consumer changes the body attribute from strong to light. The producer recalculates the similarity with the most specific set which is equal to { {*Elyse,(MerlotGrape, MalbecGrape),OffDry, Strong, Medium, Red, NapaRegion*}, { *Elyse, MalbecGrape,OffDry, Strong, Light,Red, NapaRegion* }}. At the third time, the maximum similarity is equal to 0.429 $((2 * (3/7) + 3/7)/3)$. Therefore, $P18$ is offered to the customer.

Since this offer is consistent with the customer's preferences, the consumer agent accepts the offer. When we investigate the offers, we can see that the producer agent learns that the winery is an important feature for the customer. In the third iteration, the producer finds a wine product that matches the consumer's preferences.

4.3.2. Example of ID3 system

Consider the consumer agent in Section 4.3.1 requesting *(Elyse, MerlotGrape, OffDry, Strong, Medium, Red, NapaRegion)*. We will apply the same scenario in the system using ID3 Algorithm.

After taking the first request, the producer agent adds it into its positive service list, which will be used to select the best service. Initially, every wine in the stock is a candidate for the counter offer since newly constructed decision tree has only one node labeled as positive. Among all services, the system offers the service that is the most similar one to this request. Table 4.10 shows the estimated similarities. $P3$ is selected as the most similar service to the request, so the producer agent offers this product to the consumer. The consumer agent rejects this service since the winery of this service is different from the one that consumer wishes.

After rejection of counter offer, the system add this counter offer into its negative list. After the second request (Elyse, MalbecGrape, OffDry, Strong, Medium, Red, NapaRegion), the decision tree is rebuilt again and according to tree shown in Figure 4.13, if the winery of the service is Elyse, the service is considered as positive meaning possibly acceptable service and if the winery is Forman, the service is thought as negative meaning possibly rejectable service. In that case, the system filters the available service list according to the classification of that tree. All negative services are removed from the tree. In this part, $P8$ and $P3$ are removed.

There are three services having maximum similarity. Among these services product $P2$ is selected because its rating value is higher than others (Its rating value is equal to five when that of $P5$ is equal to three and that of $P18$ is equal to four). Absolutely, $P2$ will be rejected by the consumer since the winery of that is not matched with the one consumer wishes.

After the third request, the producer rebuilds the decision tree. The constructed decision tree after the third request is shown in Figure 4.14. In this part, only the product $P2$ is eliminated from the available list. The product $P18$ seems to be the best counter offer because its average similarity to the positive service list is the highest one. When the producer offers $P18$, the consumer accepts this service supplying consumer's all conditions: winery, color and location. By coincidence, the interactions in the system using ID3 is same as that using MVS shown in Table 4.8.

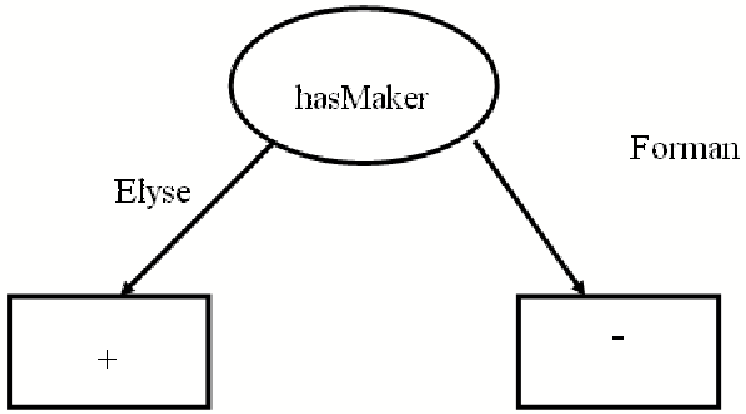

Figure 4.13. Decision tree after second request

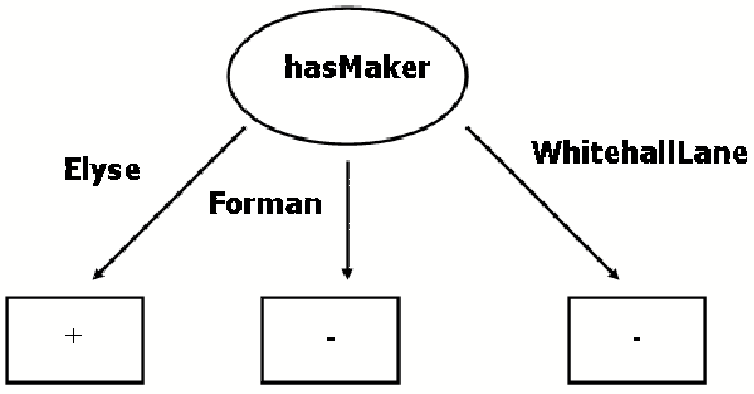

Figure 4.14. Decision tree after third request

Table 4.10. Estimated similarities for ID3 example

Product ID	After Req.1	After Req.2	After Req.3
P1	0.143	0.143	0.190
P2	0.429	**0.429**	-
P3	**0.571**	-	-
P4	0.286	0.286	0.238
P5	0.429	**0.429**	0.381
P6	0.286	0.286	0.286
P7	0.286	0.286	0.238
P8	0.143	-	-
P9	0.143	0.143	0.143
P10	0.143	0.143	0.095
P11	0.143	0.143	0.095
P12	0.0	0.0	0.0
P13	0.286	0.286	0.238
P14	0.143	0.214	0.286
P15	0.143	0.214	0.190
P16	0.143	0.143	0.143
P17	0.0	0.0	0.48
P18	0.429	**0.429**	**0.429**
P19	0.286	0.214	0.238
Max. Similarity:	**0.571**	**0.429**	**0.429**

4.3.3. Example of MVS system with RP Similarity

When the consumer agent requests *(Elyse, MerlotGrape, OffDry, Strong, Medium, Red, NapaRegion)*, the producer agent offers the most similar one to the most specific set S. After the first request, S is equal to the request. The system compares the S with each service in service list. When we compare the S with a service, we estimate the similarity of each feature separately in accordance with the RP Similarity Algorithm. The average sum of these similarities are equivalent to $SM_{(h,service)}$ in Equation 3.8 explained in Chapter 3. According to the this equation, the most similar service is found as $P3$. The details of estimation is explained in the Table 4.11. Note that, in our implementation, m is accepted as $2/3$ and n is taken as $4/7$ where m is the constant for parents and n is the constant for siblings as in Figure 3.5.

However, this service is not accepted by the consumer. After rejection, the producer agent trains the Modified Version Space with negative sample. Then, the second request comes. When the second request is received by the producer, it trains the Modified Version Space with positive sample. The new G and S is shown in the Figure 4.15. Before similarity estimation, the service list is filtered in accordance with the consistency with G. The $P3$ is removed from the list. Then, the similarity estimation is performed. As a result, $P18$ is the closest service to the most specific set S. The similarity estimation for $P18$ with S is examined in Table 4.12 and all similarities is shown in Table 4.13. Table 4.14 indicates the interaction between consumer and provider.

To sum up, we use the same scenario as in 4.3.1 with the same learning technique (MVS) but different similarity metric here. Instead of three iteration, the system accomplishes to negotiate at two iterations. Thus, we can say that the similarity metric may affect the performance of the system and semantic similarity may increase the performance. The following chapter will study the effects of similarity metrics on the performance in greater depth.

Table 4.11. RP similarity for $P3$ with S

$S = (Elyse, MerlotGrape, OffDry, Strong, Medium, Red, NapaRegion)$
$P3 = [Forman, CabernetSauvignonGrape, Dry, Strong, Medium, Red, NapaRegion]$
$sim_{winery} = 1 * (4/7) * (2/3)^2 = 0.254$
$sim_{grape} = 1 * (4/7) = 0.571$
$sim_{sugar} = 0.8$
$sim_{flavor} = 1.0$
$sim_{body} = 1.0$
$sim_{color} = 1.0$
$sim_{region} = 1.0$
$SM_{(h,service)} = 0.804$
$AVG - SM(service, S) = (0.804 * 1)/1 = 0.804$

Table 4.12. RP similarity for $P18$ with S

$S = Elyse, (MerlotGrape, MalbecGrape), OffDry, Strong, Medium, Red, NapaRegion$
$P18 = [Elyse, ZinfandelGrape, Dry, Moderate, Full, Red, NapaRegion]$
$sim_{winery} = 1.0$
$sim_{grape} = ((1 * (4/7)) + (1 * (4/7)))/2 = 0.571$
$sim_{sugar} = 0.8$
$sim_{flavor} = 0.6$
$sim_{body} = 0.6$
$sim_{color} = 1.0$
$sim_{region} = 1.0$
$SM_{(h,service)} = 0.796$
$AVG - SM(service, S) = (0.796 * 2)/2 = 0.796$

Table 4.13. Estimated similarities for MVS example with RP similarity

Product ID	After Req.1	After Req.2
P1	0.508	0.508
P2	0.746	0.746
P3	**0.804**	-
P4	0.731	0.731
P5	0.746	0.746
P6	0.685	0.685
P7	0.613	0.613
P8	0.556	0.556
P9	0.475	0.475
P10	0.506	0.506
P11	0.551	0.551
P12	0.494	0.494
P13	0.625	0.625
P14	0.528	0.559
P15	0.511	0.541
P16	0.492	0.492
P17	0.494	0.494
P18	0.796	**0.796**
P19	0.570	0.539
Max. Similarity:	**0.804**	**0.796**

Table 4.14. An example negotiation for MVS with RP Similarity

R-1	[Elyse,MerlotGrape,OffDry,Strong,Medium,Red,NapaRegion]
O-1	[Forman,CabernetSauvignonGrape,Dry,Strong,Medium,Red,NapaRegion]
R-2	[Elyse,MalbecGrape,OffDry,Strong,Medium,Red,NapaRegion]
O-2	[Elyse,ZinfandelGrape,Dry,Moderate,Full,Red,NapaRegion]

```
MOST SPECIFIC SET:
 {{(Red),(Elyse),(NapaRegion),(MerlotGrape, MalbecGrape),
     (OffDry),(Strong),(Medium)}}

MOST GENERAL SET: {
{(?)-(Red),(?),(?),(?),(?),(?),(?)}, {(?),(?)-(Forman),(?),(?),(?),(?),(?)},
{(?),(?),(?)-(NapaRegion),(?),(?),(?),(?)},
{(?),(?),(?),(?)-(CabernetSauvignonGrape),(?),(?),(?)},
{(?),(?),(?),(?),(?)-(Dry),(?),(?)}, {(?),(?),(?),(?),(?),(?)-(Strong),(?)},
{(?),(?),(?),(?),(?),(?),(?)-(Medium)}}
```

Figure 4.15. G and S after second request

5. PERFORMANCE EVALUATION

We have created a dataset for comparision purposes. The dataset contains 19 wine products that exist in the producer's inventory. We evaluate the performance of the proposed systems in respect to learning technique they used, *SMVS* (Similarity to Modified Version Space) and *DT* (ID3 Decision Tree Algorithm), by comparing them with the three other approaches. The first compared system uses original Candidate Elimination Algorithm called *SVS* (Similarity to Version Space). The second system randomly generates counter offers by picking random products for the user. We call this approach *Random Offering*. The third system we use (*SCR*) offers the most similar service to the current request by ignoring the prior requests made.

5.1. Comparision of Learning Algorithms

We apply a variety of scenarios on this dataset in order to see the performance differences among learning algorithms. Here, we use constantly Tversky's similarity measure. Furthermore, we use five test cases for this evaluation process. Each test case contains a list of preferences for the user and number of matches from the product list. With these test cases, we are interested in finding the number of iterations that are required for the producer to generate an acceptable offer for the consumer. Since the performance also depends on the initial request, we repeat our experiments with different initial requests.

Consequently, for each case, we run the algorithms five times with several variations of the initial requests. In each experiment, we count the number of iterations that need to be made to reach an agreement. We take the average of the this number in order to evaluate these systems fairly. As is customary, we test each algorithm with the same initial request.

5.1.1. Scenario 1

The customer wants to buy any wine whose sweetness degree is dry. There are 15 products in the inventory that meets this condition. Table 5.1 shows the average of five runs for all five approaches. Here, *number of iterations* represents the number of steps it takes for the customer's request to be fulfilled by the producer. Obviously, we want the number of iterations to be smal since this shows that the producer can learn the customer's preferences quickly.

From the result shown in Table 5.1, it is seen that *SMVS* score is slightly better than the *SCR* Technique. During the test, we observed that both *SMVS* and *SCR* techniques offer the same service at the beginning of the negotiation phase since the most specific set of the modified version space is equal to the initial request at the first time. However, in the following interaction their behaviors change. Furthermore, for this scenario, *Random Offering* is as good as *SMVS* since the inventory involves a large number of available services that are consistent with customer's preferences and the probability of offering a convenient service is very high. This probability is equal to 15/19 at the beginning of the negotiation and increases over time.

SVS and *DT* give the same results with *SMVS*. In fact, the differences in learning algorithm are not observed clearly in this scenario since a large number of services in the inventory meet the condition; as a result, the negotiation is performed immediately in most cases.

In order to see the performance differences between *SMVS* and *SCR* in detail, we give a comparative run-time example. Table 5.2 indicates the request-offer pairs of *SMVS* whereas Table 5.3 shows those of the *SCR* Technique.

The consumer starts with the request *(Ventana, MalbecGrape, Dry, Delicate, Light, Rose, CotesDOrRegion)*. The similarities that are estimated after each request for both *SMVS* and *SCR* are shown in Table A.1. It is seen that the similarity values are the same for both systems after the initial request. The maximum similarity is cal-

Table 5.1. Number of iterations for scenario 1

Run	SMVS	SCR	Random Offering	SVS	DT
R1	2	3	1	2	2
R2	1	1	1	1	1
R3	1	1	2	1	1
R4	1	1	1	1	1
R5	1	1	1	1	1
Average:	**1.2**	**1.4**	**1.2**	**1.2**	**1.2**

culated as 4/7, equally 0.5714. As a result, both systems offer the product $P14$. After second request, maximum similarity is estimated for SCR as 2/7, equally 0.2857. This similarity is based on only current request. Then, the product $P13$ is offered by SCR system but not accepted by the consumer agent. On the other hand, the similarity calculation for *SMVS* is based on the most specific set, which is equal to {*Ventana, MalbecGrape, Dry, Delicate,(Light, Medium), Rose, CotesDOrRegion*}. According to this calculation, the similarity of product $P19$ is equal to 3/7, whereas the similarity of product $P13$ is 2/7. Consequently, *SMVS* offers product P19 and it will be accepted by the consumer agent. The third offer will be product $P3$ for *SCR* technique. In this example, it is obvious that considering the most specific set involving all previous requests in the interaction is beneficial for offering closer services to the customer's preferences. Hence, as seen in Table 5.2 and Table 5.3, *SMVS* find a suitable offer in the second attemp whereas *SCR* takes one more step.

Table 5.2. Running of scenario 1 using SMVS

R1	(Ventana,MalbecGrape,Dry,Delicate,Light,Rose,CotesDOrRegion)
O1	P14=(CongressSprings,MalbecGrape,Sweet,Delicate,Light,Rose,NapaRegion)
R2	(Ventana,MalbecGrape,Dry,Delicate,Medium,Rose,CotesDOrRegion)
O2	P19=(ChateauMargauxWinery,MerlotGrape,Dry,Delicate,Light,Red,MargauxRegion)

Table 5.3. Running of scenario 1 using SCR

R1	(Ventana,MalbecGrape,Dry,Delicate,Light,Rose,CotesDOrRegion)
O1	P14=(CongressSprings,MalbecGrape,Sweet,Delicate,Light,Rose,NapaRegion)
R2	(Ventana,MalbecGrape,Dry,Delicate,Medium,Rose,CotesDOrRegion)
O2	P13=(Ventana,CheninBlancGrape,OffDry,Moderate,Medium,White,CentralCoastRegion)
R3	(Ventana,MalbecGrape,Dry,Strong,Medium,Rose,CotesDOrRegion)
O3	P3=(Forman,CabernetSauvignonGrape,Dry,Strong,Medium,Red,NapaRegion)

5.1.2. Scenario 2

The customer wants to buy any wine that is red and dry. There are eight products (out of 19) in the inventory that meet this condition. When we look at the results of all systems, the performances of *SMVS*, *SVS*, *DT* and *SCR* are equal and better than that of *Random Offering* (Table 5.4). At the beginning of the negotiation, the probability of offering an acceptable service for *Random Offering* is equal to 8/19. Therefore, the number of iterations is increased with respect to the first case.

Table 5.4. Number of iterations for scenario 2

Run	SMVS	SCR	Random Offering	SVS	DT (ID3)
R1	2	2	3	2	2
R2	2	2	1	2	2
R3	1	1	1	1	1
R4	1	1	3	1	1
R5	1	1	5	1	1
Average:	**1.4**	**1.4**	**2.6**	**1.4**	**1.4**

5.1.3. Scenario 3

The customer wants to buy a wine that is red, dry and moderate. There are four products meeting this condition. The test results in Table 5.5 indicate that SMVS, SVS

and DT show the best performance and the worst performance belongs to *Random Offering*. When the number of available service that matches the customer's preferences decreases, the performance difference between the proposed systems such as *SMVS* and *DT*, and other techniques becomes clear.

Table 5.5. Number of iterations for scenario 3

Run	SMVS	SCR	Random Offering	SVS	DT (ID3)
R1	2	2	3	2	2
R2	1	1	10	1	1
R3	1	1	4	1	1
R4	1	1	2	1	1
R5	2	4	3	2	2
Average:	**1.4**	**1.8**	**4.4**	**1.4**	**1.4**

According to results, the number of iteration for this scenario in *SMVS* and *SVS* is same. Although they seem to behave similarly, in most cases their behavior may be different. For instance, consider the following run for this scenario. The consumer requests *(Foxen, ChardonnayGrape, Dry, Moderate, Full, Red, SantaCruzMountainsRegion)* as seen Table 5.6 and Table 5.7.

Table 5.6. Running of scenario 3 using SMVS

R1	(Foxen,ChardonnayGrape,Dry,Moderate,Full,Red,SantaCruzMountainsRegion)
O1	P8=(Forman,ChardonnayGrape,Dry,Moderate,Full,White,NapaRegion)
R2	(ChateauMorgon,GamayGrape,Dry,Moderate,Light,Red,NapaRegion)
O2	P18=(Elyse,ZinfandelGrape,Dry,Moderate,Full,Red,NapaRegion)

Both *SMVS* and *SVS* initially offer the same service, the product $P8$. After rejection of this offer, the consumer agent wants *(ChauteauMorgon, GamayGrape, Dry, Moderate, Light, Red, NapaRegion)*. The producer agent of *SMVS* system offers the product $P18$ whereas $P2$ is offered by SVS. Despite the fact that the agents negotiate

at second time and their first counter offer is identical to each other, the last counter offer of SMVS accepted by the consumer is different from that of *SVS*. This situation stems from the generation of G and S in both system. In details, the most specific set in SVS has a form as {*(?, ?, Dry, Moderate, ?, Red, ?)*} while S in *SMVS* is equal to {{*Foxen, ChardonnayGrape, Dry, Moderate, Full, Red, SantaCruzMountainsRegion*}, {*ChateauMorgon, GamayGrape, Dry, Moderate, Light, Red, NapaRegion*}}. The most specific set of CEA algorithm is more general than that of Modified CEA. Therefore, the similarities to S of these systems will be different. As a result, they offer different services in this case. The interaction is shown in Table 5.6 and Table 5.7.

Table 5.7. Running of scenario 3 using SVS

R1	(Foxen,ChardonnayGrape,Dry,Moderate,Full,Red,SantaCruzMountainsRegion)
O1	P8=(Forman,ChardonnayGrape,Dry,Moderate,Full,White,NapaRegion)
R2	(ChateauMorgon,GamayGrape,Dry,Moderate,Light,Red,NapaRegion)
O2	P2=(WhitehallLane,CabernetFrancGrape,Dry,Moderate,Medium,Red,NapaRegion)

5.1.4. Scenario 4

The customer wants to buy a wine that is strong and red. There are only two products in the inventory that meets this condition. Again, we look at the average of five runs for five approaches. Table 5.8 shows that *SMVS* approximately finds the suitable offer at 2.2 iterations whereas *SCR* finds the convenient counter-offer at 2.8 iterations. The performance of the *Random Offering* is comparatively low. The best performance belongs to SVS with the 1.8 average iteration. DT shows nearly the same performance with 2.0 average iteration.

The reason why original Version Space shows better performance than Modified Version Space may be that the concept being learnt is in a conjuctive form and Version Space is good at learn such concepts. The most general set of VS is more specific than that of MVS so it filters rejectable services better. Also, VS generalizes its specific set much more than MVS does. Therefore, in some cases like this scenario, VS may

find the acceptable service sooner than MVS. In fact, this depends on the requests and counter offers in the interaction.

Table 5.8. Number of iterations for scenario 4

Run	SMVS	SCR	Random Offering	SVS	DT (ID3)
R1	2	4	16	2	2
R2	4	4	6	3	3
R3	1	1	11	1	1
R4	1	1	10	1	1
R5	3	4	5	2	3
Average:	**2.2**	**2.8**	**9.6**	**1.8**	**2**

5.1.5. Scenario 5

The customer wants to buy a wine whose flavor is strong and color is red or rose. There are three products meeting this condition. This case is beneficial to see the effect of learning disjunctive concepts such as (Color = Red or Rose) with the conjunctives (and Flavor = Strong). In some situations, *SCR* technique can find the convenient service sooner than *SMVS* and in some situation *SMVS* can show better performance. This depends on the requests. Average performance of *SMVS* over five runs is better than that of *SCR* as seen in Table 5.9.

Table 5.9. Number of iterations for scenario 5

Run	SMVS	SCR	Random Offering	CEA	ID3
R1	2	6	7	2	2
R2	1	1	10	1	1
R3	3	2	13	No Offer	2
R4	3	3	6	3	3
R5	1	1	2	1	1
Average:	**2**	**2.6**	**7.6**	**1.75** + no offer	**1.8**

DT is better than *SMVS*. To illustrate this, consider the consumer agent request *(ChateauMargauxWinery, MerlotGrape, Dry, Strong, Light, Red, BeaujolaisRegion)*. Both *DT* and *SMVS* offer the product $P19$. When this offer is rejected by the consumer, consumer changes her request with *(Ventana, GamayGrape, Dry, Strong, Full, Rose, GermanyRegion)* as seen in Table 5.10 and 5.11.

Table 5.10. Running of scenario 5 using SMVS - third run

R1	(ChateauMargauxWinery,MerlotGrape,Dry,Strong,Light,Red,BeaujolaisRegion)
O1	P19=(ChateauMargauxWinery,MerlotGrape,Dry,Delicate,Light,Red,MargauxRegion)
R2	(Ventana,GamayGrape,Dry,Strong,Full,Rose,GermanyRegion)
O2	P1=(ChateauMorgon,GamayGrape,Dry,Delicate,Light,Red,BeaujolaisRegion)
R3	(SchlossRothermel,GamayGrape,Dry,Strong,Full,Rose,GermanyRegion)
O3	P16=(SchlossRothermel,SangioveseGrape,Sweet,Strong,Full,Rose,GermanyRegion)

After this request, *SMVS* only eliminates the $P19$ from producer service list. On the other hand, constructed decision tree tells if the value of flavor is strong, it is a positive sample that can be accepted by the consumer and if the value of flavor is delicate, then it is a negative sample that can be rejected. Therefore, according to the algorithm, the products whose flavor value is equal to delicate, are eliminated from the service list.

As a result, $P1$, $P14$ and $P19$ are removed from the list and they would not be a counter offer in the current interaction. After the second request, *DT* offers an acceptable service whereas *SVMS* offers $P1$ which would be rejected by the consumer agent.

SVS has a good average iteration but at one run time, it cannot offer any service. To illustrate this, consider the following run. The interaction between consumer and producer is shown in Table 5.12. After rejection of first offer, the most general set turn into {*(?, ?, ?, Strong, ?, ?, ?)(?, ?, ?, ?, ?, ?, BeaujolaisRegion)*}. The services, which are not covered by this set will be removed from the service list. After filtering, only

Table 5.11. Running of scenario 5 using ID3 - third run

R1	(ChateauMargauxWinery,MerlotGrape,Dry,Strong,Light,Red,BeaujolaisRegion)
O1	P19=(ChateauMargauxWinery,MerlotGrape,Dry,Delicate,Light,Red,MargauxRegion)
R2	(Ventana,GamayGrape,Dry,Strong,Full,Rose,GermanyRegion)
O2	P6=(SantaCruzMountainVineyard,CabernetSauvignonGrape,Dry,Strong, Full,Red,SantaCruzMountainsRegion)

four services remain in service list. After the second request, the general set is equal to empty set because of the inconsistency. As discussed before, CEA algorithm does not support learning disjunctive concepts. The system unloads all services from the service list. In spite of having several services being wished by the customer, the producer cannot offer one of these services to consumer agent. This situation is undesirable in e-commerce.

Table 5.12. Running of scenario 5 using SVS

R1	(ChateauMargauxWinery,MerlotGrape,Dry,Strong,Light,Red,BeaujolaisRegion)
O1	P19=(ChateauMargauxWinery,MerlotGrape,Dry,Delicate,Light,Red,MargauxRegion)
R2	(Ventana,GamayGrape,Dry,Strong,Full,Rose,GermanyRegion)
O2	P9=(Mountadam,ChardonnayGrape,Dry,Strong,Full,White,SouthAustraliaRegion)
R3	(SchlossRothermel,GamayGrape,Dry,Strong,Full,Rose,GermanyRegion)
O3	— NO OFFER –

5.2. Comparision of Similarity Metrics

To compare the similarity metrics, we fix the learning phase with Modified Version Space. A variety of similarity metrics is applies to *SMVS* system. The scenarios discussed in Section 5.1 and we also add two different scenarios. Then, to see the performance difference among them clearly, we additionaly use a second dataset with 50 products. Again this dataset has been created for testing purposes. For this dataset, we apply three scenarios and compare the results.

5.2.1. Scenario 1

The customer wants to buy any wine whose sweetness degree is dry. There are 15 products meeting this condition. Table 5.13 shows that Wu Palmer's and our proposed similarity metric (RP) gives the best result whereas the performance of Tversky's and Lin's metrics is slightly worse. However, Resnik's similarity has the worst performance. This stems from the fact that Resnik's similarity metric does not normalize the similarity values between zero and one although others normalize those. Some features such as body, sugar and flavor have one for the maximum similarity but with Resnik's similarity measure sometimes similarity between other features such as region may have higher similarity than one. This results in unfair overweighted values for this feature. Thus, it may give worse performance since the similarity is not equally measured.

Table 5.13. Number of iterations for scenario 1 in SMVS

Run	Tversky	Resnik	Lin	Wu Palmer	RP
1	2	4	2	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	1	3	1	1	1
5	1	1	1	1	1
Average	**1.2**	**2**	**1.2**	**1**	**1**

5.2.2. Scenario 2

The customer wants to buy any wine that is red and dry. There are eight products out of 19 in the stock that meet this condition. For this scenario, using Tversky's and Lin's similarity metrics give the best result. According to Table 5.14 Wu Palmer's and RP's similarity metric are also comparable. Again, Resnik's similarity metric's performance is not as good as others.

Table 5.14. Number of iterations for scenario 2 in SMVS

Run	Tversky	Resnik	Lin	Wu Palmer	RP
1	2	6	3	3	3
2	2	2	1	1	1
3	1	1	1	1	1
4	1	2	1	2	2
5	1	3	1	1	1
Average	**1.4**	**2.8**	**1.4**	**1.6**	**1.6**

5.2.3. Scenario 3

The customer wants to buy any wine that is red, dry and moderate. Producer has four products meeting this condition. Table 5.15 indicates that the best performance is shown by Tversky's Similarity Metric. This performance is followed up by Lin's metric. Wu Palmer's metric and RP metric behave similarly at average and their performance is higher than Resnik's one.

Table 5.15. Number of iterations for scenario 3 in SMVS

Run	Tversky	Resnik	Lin	Wu Palmer	RP
1	2	5	3	3	3
2	1	2	1	1	1
3	1	1	1	1	1
4	1	2	2	3	3
5	2	2	2	2	2
Average	**1.4**	**2.4**	**1.8**	**2**	**2**

5.2.4. Scenario 4

The preference of the customer is that flavor should be strong and the color of wine should be red. There are only two products in the inventory. When we look at the results in Table 5.16, Lin's metric increases the performance much more than others

and the performance Wu Palmer's and RP metric are equal and better than Tversky's metric. According to results, Resnik's metric is not beneficial as much as others.

Table 5.16. Number of iterations for scenario 4 in SMVS

Run	Tversky	Resnik	Lin	Wu Palmer	RP
1	2	3	1	1	1
2	4	6	1	1	1
3	1	3	1	1	1
4	1	1	1	1	1
5	3	1	1	2	2
Average	**2.2**	**2.8**	**1**	**1.2**	**1.2**

5.2.5. Scenario 5

The customer wants to buy any wine whose flavor is strong and color of wine is red or rose. There are three products, which meet this condition. Table 5.17 shows that Lin's, Wu Palmer's and RP metric approximately finds the acceptable offer at 1.6 iterations while Tversky's metric finds at average 2 iterations. The performance of Resnik's metric is comparatvely low. As is customary, after this scenario, we run the algorithms three times with a variety of requests instead of five times.

Table 5.17. Number of iterations for scenario 5 in SMVS

Run	Tversky	Resnik	Lin	Wu Palmer	RP
1	2	6	1	1	1
2	1	5	1	1	1
3	3	3	3	3	3
4	3	2	2	2	2
5	1	3	1	1	1
Average	**2**	**3.8**	**1.6**	**1.6**	**1.6**

5.2.6. Scenario 6

The customer wants to buy wine whose winery is located in California (i.e. NapaRegion, CentralCostRegion) and whose grape has a type of white grape. Moreover, the winery of the wine should not be expensive. There are only four products meeting these conditions. According to Table 5.18, the behavior of Wu Palmer's metric and RP metric is similar to each other. The best performance belongs to Resnik's metric. Tversky's metric is comparatively lower than other. This scenario is a good example to see the benefits of the use of semantic similarity metrics. The metrics considering the semantic closeness increase the system performance significantly.

Table 5.18. Number of iterations for scenario 6 in SMVS

Run	Tversky	Resnik	Lin	Wu Palmer	RP
1	2	1	1	1	1
2	2	3	5	4	4
3	9	3	5	3	3
Average	**4.3**	**2.3**	**3.7**	**2.7**	**2.7**

As said before, this situation stems from the lack of normalization in Resnik's metric. Some similarity values are higher than one. However, we use normalized similarity values for some featues such as body. Using both metrics in the same estimation to decide the best offer is not fair. Therefore, we do not include this metric in our comparision for the next scenarios.

5.2.7. Scenario 7

The customer wants to buy wine whose color is red or rose and grape type is red grape. In addition, the location of wine should be in Europe. The sweetness degree is wished to be dry or offdry. And flavor would be delicate or moderate where the body should be medium or light. Furthermore, the winery of the wine should be an expensive winery. There is only one product meeting all these requirements, product

$P19$. As it seen from Table 5.19, Wu Palmer's and our similarity metric RP obtain the best performance, whereas Tversky's performance is comparatively lower than all metrics since it ignores the semantic closeness of features.

Table 5.19. Number of iterations for scenario 7 in SMVS

Run	Tversky	Resnik	Lin	Wu Palmer	RP
1	4	3	2	2	2
2	12	3	2	1	1
3	2	2	1	1	1
Average	**6**	**2.7**	**1.7**	**1.3**	**1.3**

5.2.8. Scenario 8 with Second Dataset

We use the second dataset to see the performance differences better. Also, we do not investigate the Resnik's similarity metrics any more. The customer wants to buy moderate rose wine, which is located in around French Region. The category of winery should be Moderate Winery. According to the Table 5.20, the performance of Lin's metric and Wu Palmer's metric is equal and better than our proposed similarity metric (RP). The worst performance belongs to Tversky's metric again.

Table 5.20. Number of iterations for scenario 8 in SMVS

Run	Tversky	Lin	Wu Palmer	RP
1	11	2	2	2
2	8	4	3	4
3	3	2	3	3
Average	**7.3**	**2.7**	**2.7**	**3**

5.2.9. Scenario 9 with Second Dataset

The customer wants to buy expensive red wine, which is located around California Region or cheap white wine, which is located in around Texas Region. Available

products are $P2$, $P6$, $P13$ and $P50$. Table 5.21 indicates that the systems using RP and Wu Palmer's similarity metric find out the appropriate products at average 2 iterations. Similar to previous scenario, the performance of Tversky's similarity metric is worse than other semantic similarity measures.

Table 5.21. Number of iterations for scenario 9 in SMVS

Run	Tversky	Lin	Wu Palmer	RP
1	14	7	2	2
2	1	1	1	1
3	5	4	3	3
Average	**6.7**	**4**	**2**	**2**

5.2.10. Scenario 10 with Second Dataset

The customer wants to buy delicate white wine whose producer in the category of Expensive Winery. The available products are $P42$ and $P44$. According to the result, RP similarity metric has better performance than others. Table 5.22 indicates that the performance of Lin's and Wu Palmer's metric is nearly same. The worst performance belongs to Tversky's metric.

Table 5.22. Number of iterations for scenario 10 in SMVS

Run	Tversky	Lin	Wu Palmer	RP
1	2	2	4	2
2	5	2	1	2
3	8	4	3	3
Average	**5**	**2.7**	**2.7**	**2.3**

5.3. Results and Discussion

Table 5.23 compares a variety of approaches using different learning algorithm. SVS gives the best results but it seems that it is not robust for the cases containing

disjuctive preferences. At any time, it may not generate a counter offer to the consumer. ID3 and SMVS give comparable results with SVS and they are more robost to disjuctive preferences. When the number of possible services is decreasing, the time interval for offering an acceptable service gets longer in Random Offering. When the large parts of inventory is compatible with the customer's preferences as in the first test case, the performance of all techniques are nearly same.

Table 5.23. Average number of iterations in comparision of learning algorithms

Run	SMVS	SCR	Random Offering	SVS	DT
Scenario 1:	1.2	1.4	1.2	1.2	1.2
Scenario 2:	1.4	1.4	2.6	1.4	1.4
Scenario 3:	1.4	1.8	4.4	1.4	1.4
Scenario 4:	2.2	2.8	9.6	1.8	2
Scenario 5:	2	2.6	7.6	1.75+ No offer	1.8
Average of all cases:	**1.64**	**2**	**5.08**	**1.51**+No offer	**1.56**

We compare similarity metrics with SMVS system. Table 5.24 indicates that Wu Palmer's metric and RP similarity measure nearly give the same performance and better than others. When the results are examined, considering semantic closeness increases the performance. Tversky's metric ignoring semantics give the worst performance. The behavior of Resnik's and Lin's are nearly similar.

Table 5.24. Average number of iterations in comparision of similarity metrics

Run	Tversky	Resnik	Lin	Wu Palmer	RP
Scenario 1:	1.2	2	1.2	1	1
Scenario 2:	1.4	2.8	1.4	1.6	1.6
Scenario 3:	1.4	2.4	1.8	2	2
Scenario 4:	2.2	2.8	1	1.2	1.2
Scenario 5:	2	3.8	1.6	1.6	1.6
Scenario 6:	4.3	2.3	3.7	2.7	2.7
Scenario 7:	6	2.7	1.7	1.3	1.3
Scenario 8:	7.3	-	2.7	2.7	3
Scenario 9:	6.7	-	4	2	2
Scenario 10:	5	-	2.7	2.7	2.3
Average of all cases:	**3.75**	**2.69**	**2.18**	**1.88**	**1.87**

6. CONCLUSION

As an alternative to price-oriented negotiation approaches, we propose an approach based on the content of the services, where the producers learn the preferences of consumers over time. The aim is to automate the negotiation process and to provide the best service that is most similar to the one desired by the consumer.

Without doubt, offering a service by considering the preferences of the consumer will improve service quality by shortening the negotiation time. Mostly, both agents have limited information about the other agent at the beginning of the interaction. However, they can learn some required information related to the negotiation over the time by using machine learning techniques. In a negotation process, the most important issue is the consumer's preferences as far as service quality is concerned. Thus, the producer models the requests of the consumer and his counter offers to understand which features are more important for the consumer. The proposed architecture for the negotiation combines important ideas from incremental learning techniques with expressive representation of ontologies.

To accomplish this, applying Candidate Elimination Algorithm seems a good solution. Unfortunately, this algorithm does not support learning the preference in a disjunctive form. It only learns the conjuntive concepts. Therefore, we improve the algorithm support disjunctions. Firstly, we extend the hypothesis language and make it hold all possible target concepts. In detail, we specialize the most general set minimally and generalize the most specific set minimally as possible. This general set tries to lose as minimal information as possible. Moreover, the most general set is used to filter the possible service list from the services that will possibly be rejected by the consumer. To accomplish it, the algorithm eliminates the services that are not covered by the most general set. In addition, the most specific set is used to offer the best service that is most similar to consumer's request. When the producer decides which service in his inventory mostly satisfies the needs of the consumer as far as the preferences are considered, it estimates the similarity for each service in his inventory

to the most specific set.

One of the advantages of the modified version of Candidate Elimination Algorithm is that it can learn the disjuntive concepts. However, the general set grows exponentially. As the behaviour of MVS is observed, we suspect that the performance of the MVS without a general set would be similar to the performance of MVS as it is, due to the fact that general set covers all possible services and minimally specialized with the negative examples. As a result, only the rejected service is removed from the service list. Accordingly, a better result can be obtained if the most general set is specialized in a different way.

As an alternative to Version Space based learning, decision trees can be used. We implement the ID3 algorithm in order to learn the preferences. The tree structure can learn both conjunctions and disjunctions. Since decision tree acts as a classifier, the services in the service list that is classified as negative by the decision tree are eliminated from the service list, the remaining services are tested. The most similar one is offered to the consumer. In fact, ID3 is not an incremental learning method. However, we rebuild the tree after each request. Thus, using ID3 iteratively behaves as an incremental algorithm. Of course, this requires additional storage since all the request and counter offer that is rejected by the consumer are needed to keep. In addition, it requires additional process to build the tree from stratch.

In fact, there are some incremental algorithms for the decision trees such as IDR5. Since, in this algorithm the complexity goes up with the number of features and their possible values, it can be acceptable to use ID3 iteratively by rebuilding the decision tree at each coming request [45].

Moreover, ontologies are used as a communication device for the producer and the consumer. By the help of a shared ontology, both agents use the same vocabulary free from the ambiguity. In addition, the agents obtain useful information being used in negotiation by reasoning by using the ontology. The taxonomies can be represented with the ontologies and they can be used to estimate the semantic distance of two

concepts. To take this advantage of the ontologies, we use semantic similarity metrics while estimating the similarity between the learnt target service and the available services in the producer's inventory.

Another issue related to the negotiation is similarity estimation. We propose to use semantic closeness in generation of the counter offer. Also, a new similarity metric based on the taxonomy is proposed. Its performance is as well as other semantic metrics. An important finding of this thesis is using semantic closeness with learning increase the performance of the negotiation.

We use OWL [24] as our ontology language and JENA2 as our ontology reasoner. In fact, firstly we have used KAON2 [46] instead of JENA2 [47]; nevertheless, KAON2 does not support reasoning with nominal ontologies. As mentioned before, the system uses the well-known *Wine Ontology* with some extensions and modifications. This ontology involves some owl forms such as "owl:oneOf class" and "owl:hasValue", which are indicators for nominal concepts. Therefore, the final system is developed by using Jena2 Ontology Reasoner. The *Wine Ontology* is validated via an ontology validator [48] after modifications and extensions.

Our approach is open for improvements. First, the preferences of the consumer agent may change during the negotiation. It would be interesting to extend the learning algorithm to deal with dynamic changes in the requests of the consumer agent. The preferences can be enriched with subtle relations on price.

We plan to integrate ontology reasoning into the learning algorithm so that hierarchical information can be learned from subsumption hierarchy object or subclass of relations. Further, by using relationships among features, the producer can discover new knowledge from the existing knowledge.

Currently, the producer only tries to learn the consumer's demand without considering its own preferences. Incorporating the business strategies of the producer will allow more realistic scenarios to be tested. These are interesting directions that we will

pursue in our future work.

APPENDIX A: SIMILARITY ESTIMATION

Table A.1. The estimated similarities for scenario 1

	Similarities for SMVS		Similarities for SCR		
ID	Request1	Request2	Request1	Request2	Request3
P1	0.4286	0.4286	0.4286	0.2857	0.1429
P2	0.1429	0.2857	0.1429	0.2857	0.2857
P3	0.1429	0.2857	0.1429	0.2857	**0.4286**
P4	0.1429	0.2857	0.1429	0.2857	0.2857
P5	0.1429	0.2857	0.1429	0.2857	0.2857
P6	0.1429	0.1429	0.1429	0.1429	0.2857
P7	0.1429	0.2857	0.1429	0.2857	0.2857
P8	0.1429	0.1429	0.1429	0.1429	0.1429
P9	0.1429	0.1429	0.1429	0.14295	0.2857
P10	0.1429	0.2857	0.1429	0.2857	0.2857
P11	0.1429	0.2857	0.1429	0.2857	0.2857
P12	0.1429	0.1429	0.1429	0.1429	0.1429
P13	0.1429	0.2857	0.1429	**0.2857**	-
P14	**0.5714**	-	**0.5714**	-	-
P15	0.1429	0.2857	0.1429	0.2857	0.2857
P16	0.1429	0.1429	0.1429	0.1429	0.2857
P17	0.2857	0.2857	0.2857	0.1429	0.1429
P18	0.1429	0.1429	0.1429	0.1429	0.1429
P19	0.4286	**0.4286**	0.4286	0.2857	0.1429
Max. Sim:	**0.5714**	**0.4286**	**0.5714**	**0.2857**	**0.4286**

APPENDIX B: DATASETS

Table B.1. Available services in first ontology

ID	R	Products
P1	4	(ChateauMorgon,GamayGrape,Dry,Delicate,Light,Red,BeaujolaisRegion)
P2	5	(WhitehallLane,CabernetFrancGrape,Dry,Moderate,Medium,Red,NapaRegion)
P3	4	(Forman,CabernetSauvignonGrape,Dry,Strong,Medium,Red,NapaRegion)
P4	4	(Marietta,CabernetSauvignonGrape,Dry,Moderate,Medium,Red,SonomaRegion)
P5	3	(PageMillWinery,CabernetSauvignonGrape,Dry,Moderate,Medium,Red,NapaRegion)
P6	4	(SantaCruzMountainVineyard,CabernetSauvignonGrape,Dry,Strong, Full,Red,SantaCruzMountainsRegion)
P7	2	(Bancroft,ChardonnayGrape,Dry,Moderate,Medium,White,NapaRegion)
P8	5	(Forman,ChardonnayGrape,Dry,Moderate,Full,White,NapaRegion)
P9	4	(Mountadam,ChardonnayGrape,Dry,Strong,Full,White,SouthAustraliaRegion)
P10	4	(MountEdenVineyard,ChardonnayGrape,Dry,Moderate,Medium,White,EdnaValleyRegion)
P11	3	(PeterMccoy,ChardonnayGrape,Dry,Moderate,Medium,White,SonomaRegion)
P12	4	(Foxen,CheninBlancGrape,Dry,Moderate,Full,White,SantaBarbaraRegion)
P13	5	(Ventana,CheninBlancGrape,OffDry,Moderate,Medium,White,CentralCoastRegion)
P14	5	(CongressSprings,MalbecGrape,Sweet,Delicate,Light,Rose,NapaRegion)
P15	4	(Selaks,MalbecGrape,Sweet,Moderate,Medium,White,NewZealandRegion)
P16	3	(SchlossRothermel,SangioveseGrape,Sweet,Strong,Full,Rose,GermanyRegion)
P17	3	(StGenevieve,GamayGrape,Dry,Moderate,Light,White,CentralTexasRegion)
P18	4	(Elyse,ZinfandelGrape,Dry,Moderate,Full,Red,NapaRegion)
P19	4	(ChateauMargauxWinery,MerlotGrape,Dry,Delicate,Light,Red,MargauxRegion)

Table B.2. Additional available services in second ontology

ID	R	Products
P20	5	(Elyse,ZinfandelGrape,Dry,Moderate,Full,Red,USRegion)
P21	2	(GarryFarrel,PinotNoirGrape,Sweet,Moderate,Light,Rose,USRegion)
P22	4	(LaneTanner,PetiteSyrahGrape,Dry,Strong,Medium,Red,BordeauxRegion)
P23	3	Beringer,PetiteVerdotGrape,OffDry,Moderate,Medium,Red,BordeauxRegion)
P24	4	(ChateauMorgon,GamayGrape,OffDry,Moderate,Medium,Rose,BordeauxRegion)
P25	5	StGenevieve,CabernetFrancGrape,Dry,Strong,Light,Red,MendocinoRegion)
P26	4	(StGenevieve,PinotNoirGrape,Sweet,Delicate,Light,Rose,PauillacRegion)
P27	2	(Handley,PinotBlancGrape,OffDry,Moderate,Medium,Rose,LoireRegion)
P28	3	(KathrynKennedy,GamayGrape,Dry,Delicate,Light,Red,BeaujolaisRegion)
P29	5	(Handley,GamayGrape,Dry,Delicate,Light,Red,BeaujolaisRegion)
P30	3	(Elyse,MerlotGrape,Dry,Moderate,Medium,Red,PortugalRegion)
P31	3	(Taylor,CheninBlancGrape,OffDry,Moderate,Full,White,LoireRegion)
P32	4	(WhitehallLane,RieslingGrape,Sweet,Strong,Light,White,LoireRegion)
P33	4	(Ventana,SangioveseGrape,Sweet,Moderate,Full,Rose,GermanyRegion)
P34	5	(Beringer,RieslingGrape,OffDry,Strong,Light,Rose,GermanyRegion)
P35	5	(ChateauMorgon,CheninBlancGrape,Dry,Strong,Light,White,BourgogneRegion)
P36	4	(PeterMccoy,MalbecGrape,Sweet,Delicate,Medium,Rose,AlsaceRegion)
P37	3	(PageMillWinery,PetiteVerdotGrape,Dry,Moderate,Full,Red,SancerreRegion)
P38	2	(McGuinnesso,CabernetSauvignonGrape,Sweet,Moderate,Light,Red,FrenchRegion)
P39	3	(Beringer,PetiteSyrahGrape,Dry,Delicate,Full,Red,MendocinoRegion)
P40	5	(Marietta,ZinfandelGrape,Dry,Moderate,Light,Red,NewZealandRegion)
P41	4	(Forman,SangioveseGrape,Dry,Strong,Medium,Red,MuscadetRegion)
P42	4	(ClosDeLaPoussie,SauvignonBlancGrape,Dry,Delicate,Light,White,BourgogneRegion)
P43	5	(GaryFarrell,SangioveseGrape,Dry,Moderate,Medium,Red,ChiantiRegion)
P44	3	(SantaCruzMountainVineyard,SemillonGrape,Sweet,Delicate,Light,White,SantaBarbaraRegion)
P45	3	(LaneTanner,PinotBlancGrape,OffDry,Strong,Full,White,MedocRegion)
P46	4	(CongressSprings,SauvignonBlancGrape,OffDry,Strong,Medium,White,AlsaceRegion)
P47	4	(Bancroft,SangioveseGrape,OffDry,Moderate,Medium,Rose,EdnaValleyRegion)
P48	5	(Forman,PinotBlancGrape,Dry,Strong,Light,White,BourgogneRegion)
P49	5	(StGenevieve,MalbecGrape,OffDry,Moderate,Full,Red,MendocinoRegion)
P50	4	(Ventana,SemillonGrape,Sweet,Delicate,Light,White,TexasRegion)

REFERENCES

1. Aydoğan, R. and P. Yolum, "Learning Consumer Preferences for Content-Oriented Negotiation", *AAMAS Workshop on Business Agents and the Semantic Web (BASeWEB)*, pp. 43–52, Hakodate, Japan, May 2006.

2. Singh, M. P., "Value-oriented Electronic Commerce", *IEEE Internet Computing*, Vol. 3, No. 3, pp. 6–7, 1999.

3. Maes, P., R. H. Guttman and A. G. Moukas, "Agents that Buy and Sell", *Communications of the ACM*, Vol. 42, No. 3, pp. 81–91, 1999.

4. Fatima, S., M. Wooldridge and N. Jennings, "Optimal Agents for Multi-issue Negotiation", In *Proceeding of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 129–136, ACM Press, Melbourne, Australia, July 2003.

5. Busch, L. and I. Horstman, "A Comment on Issue-by-issue Negotiations", *Games and Economic Beh.*, Vol. 19, No. 1, pp. 144–148, 1997.

6. Faratin, P., C. Sierra and N. R. Jennings, "Using Similarity Criteria to Make Issue Trade-offs in Automated Negotiations", *Artificial Intelligence*, Vol. 142, No. 2, pp. 205–237, Elsevier, 2002.

7. Giraud-Carrier, C., "A Note on the Utility of Incremental Learning", *AI Communications*, Vol. 13, No. 4, pp. 215–223, 2000.

8. Mitchell, T. M., "Generalization as Search", *Artificial Intelligence*, Vol. 18, No. 2, pp. 203–226, 1982.

9. Mitchell, T. M., *Machine Learning*, McGraw Hill, NY, 1997.

10. Utgoff, P. E., "Incremental Induction of Decision Trees", *Machine Learning*, Vol.

4, pp. 161-186, Kluwer Academic Publisher, 1989.

11. Daconta, M. C. , L. J. Obrst and K. T. Smith, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, Wiley, Indianapolis, 2003.

12. Fensel, D., J. Hendler, H. Lieberman and W. Wahlster, *Spinning the Semantic Web*, MIT Press, Cambridge, 2003.

13. Berners-Lee, T., *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*, Harper San Francisco, USA, 1999.

14. Jasper, R. and M. Uschold, "Enabling Task-Centered Knowledge Support though Semantic Markup", In *Spinning the Semantic Web*, pp. 223-251, MIT Press, Cambridge, 2003.

15. Lassila, O. and M. Adler, "Ubiquitous Computing Meets the Semantic Web", In *Spinning the Semantic Web*, pp. 363-376, MIT Press, Cambridge, 2003.

16. McGuinness, D. L., "Ontologies Come of Age", In *Spinning the Semantic Web*, pp. 171-194, MIT Press, Cambridge, 2003.

17. Gruber, T. R., "A Translation Approach to Portable Ontologies", *Knowledge Acqusition*,Vol. 5, No. 2, pp. 199–220, 1993.

18. Heflin, J., J. Hendler, and S. Luke, "SHOE: A Blueprint for the Semantic Web" In *Spinning the Semantic Web*, pp. 29-63, MIT Press, Cambridge, 2003.

19. DAML, http://www.daml.org/about.html, 2003.

20. Horrocks, I., D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, and R. Studer, *OIL: The Ontology Inference Layer*, http://www.ontoknowledge.com/oil, Vrije Universiteit Amsterdam, September, 2000.

21. RDF: Resource Description Framework , http://www.w3.org/RDF, 2006.

22. Singh, M. P. and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*, John Willey & Sons, Ltd, England, 2005.

23. RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema, 2004.

24. OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide, 2004.

25. SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/, 2006.

26. Inductive Learning, http://www.cs.cf.ac.uk/Dave/AI2/node144.html, 2006.

27. Alpaydın, E., *Introduction to Machine Learning*, The MIT Press, October 2004.

28. Quinlan, J. R., "Induction of Decision Trees", *Machine Learning*, Vol. 1, No. 1, pp. 81-106, 1986.

29. Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., California, 1993.

30. Debenham, J. K., "Managing E-market Negotiation in Context with a Multiagent System", In *Proceedings Twenty First International Conference on Knowledge Based Systems and Applied Artificial Intelligence, ES'2002: Applications and Innovations in Expert Systems X*, Cambridge, UK, 2002.

31. Tamma, V., S. Phelps, I. Dickinson and M. Wooldridge, "Ontologies for Supporting Negotiation in E-commerce", *Engineering Applications of Artificial Intelligence*, Vol. 18, pp. 223–236, 2005.

32. Broens, T. , S. Pokraev, M. Van Sinderen, J. Koolwaaij and P. D. Costa, "Context-aware, Ontology-based Service Discovery", In *EUSAI, Lecture Notes in Computer*

Science, Springer, Vol. 3295, pp. 72–83, 2004.

33. Sadri, F., F. Toni and P. Torroni, "Dialogues for Negotiation: Agent Varieties and Dialogue Sequences", *In: Intelligent Agents VIII: 8th In-ternational Workshop, ATAL 2001, Seattle, WA, USA, Revised Papers, LectureNotes in Artificial Intelligence*, Springer-Verlag, Vol. 2333, pp. 405–421, 2001.

34. Brandt, F. and G. Weiβ, "Antisocial Agent and Vickrey Auctions", *In: Intelligent Agents VIII: 8th In-ternational Workshop, ATAL 2001, Seattle, WA, USA, Revised Papers, LectureNotes in Artificial Intelligence*,Springer-Verlag, Vol. 2333, pp. 335–347, 2001.

35. Brzostowski, J. and R. Kowalczyk, "On Possibilistic Case-based Reasoning for Selecting Partners for Multi-attribute Agent Negotiation", In *Proc. of the 4th Intl. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 273–278, July 2005.

36. Wine Ontology, http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine.rdf, 2003.

37. Pal, S. K. and S. C. K. Shiu, *Foundations of Soft Case-Based Reasoning*, John Wiley & Sons, New Jersey, 2004.

38. Tversky, A., "Features of Similarity", *Psychological Review*, Vol. 84, No. 4, pp. 327–352, 1977.

39. Resnik, P., "Using Information Content to Evaluate Semantic Similarity in a Taxonomy", *In Proceedings of IJCAI-95*, pp. 448-453, Montreal, Canada, 1995.

40. Lin, D., "An Information-theoretic Definition of Similarity", In *J. Shavlik, editor, Proc. 15th Intl. Conference on Machine Learning*, Morgan Kaufmann, pp. 296–304, San Francisco, CA, 1998.

41. Wu, Z. and M. Palmer, "Verb Semantics and Lexical Selection", *In Proceedings*

of the 32nd Annual Meeting of the Associations for Computational Linguistics, pp. 133-138, Las Cruces, New Mexico, 1994.

42. Defining N-ary Relations on the Semantic Web: Use with Individuals, http://www.w3.org/TR/2004/WD-swbp-n-aryRelations-20040721, 2004.

43. Wine Basics, http://www.wineprofessor.com/winebasics.html, 2006.

44. Oxford University Wine Society, http://users.ox.ac.uk/ bacchus/information.html, 2006.

45. Grieser, G., "Teil 5: Incremental Learning and Concept Drift (V. 1.0)" http://www.ke.informatik.tu-darmstadt.de/lehre/ws06/mldm/dt-incremental.pdf, 2006.

46. KAON2, http://kaon2.semanticweb.org, 2003.

47. Jena, http://jena.sourceforge.net/, 2006.

48. WonderWeb OWL Ontology Validator, http://phoebus.cs.man.ac.uk:9999/OWL/ Validator, 2003.