Semantic WS-Agreement Partner Selection

Nicole Oldham
LSDIS Lab
Computer Science
University of Georgia
Athens, GA, USA
oldham@cs.uga.edu

Kunal Verma
LSDIS Lab
Computer Science
University of Georgia
Athens, GA, USA
verma@cs.uga.edu

Amit Sheth
LSDIS Lab
Computer Science
University of Georgia
Athens, GA, USA
amit@cs.uga.edu

Farshad Hakimpour
LSDIS Lab
Computer Science
University of Georgia
Athens, GA, USA
farshad@hakimpour.com

ABSTRACT

In a dynamic service oriented environment it is desirable for service consumers and providers to offer and obtain guarantees regarding their capabilities and requirements. WS-Agreement defines a language and protocol for establishing agreements between two parties. The agreements are complex and expressive to the extent that the manual matching of these agreements would be expensive both in time and resources. It is essential to develop a method for matching agreements automatically. This work presents the framework and implementation of an innovative tool for the matching providers and consumers based on WS-Agreements. The approach utilizes Semantic Web technologies to achieve rich and accurate matches. A key feature is the novel and flexible approach for achieving user personalized matches.

Categories and Subject Descriptors
H.3.3 **[Information Systems]** Information Search and Retrieval

General Terms: Algorithms, Documentation, Design, Experimentation

Keywords: WS-Agreement, Semantic Policy Matching, Ontologies, OWL, ARL, Snobase, Agreement Matching, Semantic Web Service, WSDL-S, dynamic service selection, multi-ontology service annotation

1. INTRODUCTION

In a service oriented environment it is advantageous for service consumers and providers to obtain guarantees regarding the services that they both require and offer. Usually these guarantees pertain to quality of service (QoS) aspects. WSDL does not provide a means to express these guarantees; therefore such standards as WS-Policy [23] and WSLA [25] exist to allow for the expression of additional nonfunctional attributes. However, these standards are not expressive enough to represent the truly complex nature of the relationship between a service consumer and provider. The WS-Agreement specification [2] defines a language and protocol for capturing this intricate relationship with agreements between two parties. An agreement between a service consumer and a service provider specifies one or more service level objectives (SLO) which state the requirements and capabilities of each party on the availability of resources and service qualities. For example, an agreement may provide assurances on the bounds of service response time, service availability, or service reliability. WS-Agreement is more expressive than the previous policy standards because in addition to service level objectives, an agreement contains scopes for which the guarantee holds, conditions which must exist in order for the guarantee on the SLO to be valid, and business values, such as penalties and rewards, which incur if the SLO is not satisfied. This is further complicated by the *symmetry* of these agreements such that each provider does not only state guarantees regarding capabilities but likely has requirements of its own. In addition, each agreement may contain multiple alternatives of guarantee sets. As each consumer seeking a suitable provider has many complex options to choose from, the manual selection of providers is time consuming, tedious, and error prone. With the increasing acceptance and popularity of WS-Agreement and the ever present need to protect the quality of service with guarantees, the development of an approach for the automatic matching of these agreements is imperative.

This paper defines and provides reasoning methods for the components of an agreement which must be compatible for quality matches. We present a powerful approach which uses OWL ontologies to represent domain knowledge in conjunction with SWRL rules to achieve the most accurate and consumer personalized matches. The contributions of this work include:

- Creating and implementing a framework for automated matching of provider and consumer agreements that eliminates tedious and error prone manual matching.
- Use of multiple ontologies, both domain specific and domain independent for representing semantic information used by the agreements
- Presenting a flexible approach for specifying and reasoning over user defined preferences which allows the customized matching without changing matching code or possessing programming knowledge.

The remainder of this paper is organized as follows. Section 2 presents the motivation for our approach. Section 3 briefly covers the WS-Agreement schema and the general process of WS-Agreement matching. Section 4 is composed of details on our framework and implementation of Semantic WS-Agreement Partner Selection (SWAPS). Section 5 presents a real world situation which would benefit from the use of WS-Agreements and illustrates the necessity of an efficient tool for matching consumers with providers. Section 6 discusses related work, and Section 7 provides conclusions and future work.

2. MOTIVATION FOR A SEMANTIC APPROACH

The current WS-Agreement specification is based on XML based domain vocabularies and therefore limits the ability of matching the agreements to syntactical matching. Our approach proposes using domain knowledge captured using ontologies and rules to extend the matching capabilities beyond simple string matching. A matcher considering only the syntax of the agreements without the domain knowledge may not able to correctly identify all matches. We illustrate the usefulness of our approach with the

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

following example. Consider that a service consumer has the following requirement:

- *Availability is greater than 95%*

and a provider is offering the assurances:

- *Mean Time to Recover equals 5 minutes*
- *Mean Time between failures equals 15 hours*

A syntactic matcher would perform a string comparison to determine if the provider can satisfy the consumer's request. The syntactic matcher would generally determine that these two services do not match on the grounds that the provider does not provide an assurance for availability. However, our approach utilizes an ontology which provides a deeper understanding of the domain with the help of domain rules. For example, with respect to the above case:

- *Availability = Mean Time Between Failures/(Mean Time Between Failures + Mean Time To Recover)*

Therefore the semantic approach reasons that the provider is actually offering the assurance:

- *Availability equals 99.4%.*

Our matcher determines that this provider does in fact satisfy the requirements of the consumer. This example illustrates how incorporating using domain knowledge helps matching yield much more accurate matches.

3. WS-AGREEMENT AND WS-AGREEMENT MATCHING

This section briefly describes the WS-Agreement schema [2], the extensions that have been added for this work, and a general overview of the significant elements of WS-Agreement matching.

3.1 WS-Agreement Schema

WS-Agreement offers a rich language for stating the assurances and requirements of Web Services. This allows capturing and representing the complicated nature of real world agreements with the help of service level objectives (SLOs), qualifying conditions and business values. SLOs represent some capability or requirement of a provider or consumer. For example, the consumer may require that all response times be less than 5 seconds. However, in a real world environment these capabilities and requirements cannot be guaranteed under every circumstance. For instance, a service might only be able to process a job in less than 5 seconds if the number of requests at that moment is less than a thousand. Such conditions can be associated with SLOs with the help of qualifying conditions. Business values help in representing the importance, penalties, and rewards associated with SLOs.

WS-Agreements are written in XML and consist of alternative sets of guarantees denoted with the "ExactlyOne" and "ALL" tags. Due to the already complex nature of agreements, we save the WS-Agreement's "OneOrMore" tag for future work and assume that all agreements are written as a disjunction of alternative sets of guarantees. The guarantees are expressed within the "GuaranteeTerm" tag and assert assurances or requirements on the quality associated with the service. Below is the Guarantee Term schema followed by Table 1 which describes the components of a guarantee term.

```
<wsag:GuaranteeTerm Obligated="…">
  <wsag:ServiceScope ServiceName="…">…
  </wsag:ServiceScope>*
  <wsag:ServiceLevelObjective> …
  </wsag:ServiceLevelObjective>
  <wsag:QualifyingCondition>…</wsag:QualifyingCondition>?
  <wsag:BusinessValueList>…</wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

Table 1. GuaranteeTerm Components

WS-Agreement Tag	Purpose
Obligated	States the party responsible for the fulfillment of the guarantee. Value will be ServiceProvider or ServiceConsumer
ServiceScope	Describes to what service element specifically a service applies.
ServiceLevelObjective (SLO)	An assertion over the terms of the agreement which represents the QoS aspect of the agreement. Usually defines bounds usually over QoS concepts such as response time, fault rate or cost.
QualifyingCondition	Optional condition which must exist in order for the SLO to be satisfied. Usually over external factors such as time of day.
BusinessValueList	Optional values which represent the strength of commitment by stating penalties, rewards and importance

3.2 WS-Agreement Extensions

In order to achieve effective semantic matches, we extend the original WS-Agreement schema with several additional tags. The new tags allow for the incorporation of semantics into WS-Agreement and add additional structure for clarity during parsing and matching.

3.2.1 Adding Structure to SLO and Qualifying Conditions

The WS-Agreement specification was written with flexibility as one of the key goals and therefore lacks some structure in important areas such as the SLO and QualifyingCondition. The values within each of those tags can contain any possible expression. While this would be acceptable for an agreement which is intended to be read by a human user, additional structure must be added to the expressions in order to for a machine to automatically parse and reason over agreements. However, we added this structure while still preserving much of the flexibility specifically for domain specific predicates. For structure, we have added the expression, predicate, parameter, and value tags, as defined in the WSLA specification [25]. In addition there are the optional tags for unit and percent. Percent is used when a service level objective uses a percentage. For example, 99% of responseTimes are less than 5 seconds. Table 2 shows an example using the original schema as defined in [2] which is too ambiguous to parse and reason over. Our modified schema which adds structure is also shown in Table 2.

3.2.2 Adding Semantics to the WS-Agreement

Agreements contain ambiguities which we clarify using an OntConcept annotation tag. In the original schema of the Terms section of the WS-Agreement. Even though these variables have already been defined within the service description terms, it is unclear to what the summation of duration1 and duration2 actually refers. For example, it may refer to a QoS parameter responseTime or it may refer to a domain concept processOrderDuration. The addition of the OntConcept tag removes this ambiguity by linking this expression parameter directly to the concrete ontology concept. The value of OntConcept will be a concept from an ontology regardless of what variables are named and to what they refer. The agreement creators are required to include this tag to allow for semantic reasoning over the expression. This yields more effective matches than purely syntactic methods. The OntConcept tag clarifies the QoS or domain specific parameter to which the objective pertains. Figure 2 contains an example illustrating how the OntConcept tag remedies ambiguity. A syntactic matcher is not able to determine that (duration1+duration2) and processTime each refer to the response time of the service which is the concept responseTime in the QoS ontology. Adding OntConcept allows the matcher to recognize that although the concepts are syntactically different, they are semantically the same.

Table 2. SWAPS extensions to the WS-Agreement schema

	WS-Agreement Schema	**SWAPS Schema**
SLO	`<ServiceLevelObjective>` duration1+duration2 < 5 s `</ServiceLevelObjective>`	`<ServiceLevelObjective>` `<Expression>` `<Predicate type="less">` `<Parameter>`duration1+duration2 `< /Parameter>` `<OntConcept>`qos::responseTime `</OntConcept>` `<Value>`5`</Value>` `<Unit>`time:seconds`</Unit>` `</Predicate></Expression>` `</ServiceLevelObjective>`
Qualifying Condition	`<QualifyingCondition>` day of week is a weekday `</QualifyingCondition>`	`<QualifyingCondition>` `<Expression>` `<Predicate type="equals">` `<Parameter>`dayOfWeek `</Parameter>` `<OntConcept>`time:dayOfWeek `</OntConcept>` `<Value>`time:weekday`</Value>` `</Predicate></Expression>` `</QualifyingCondition>`

3.2.3 Domain Specific Predicate Flexibility

When extending the WS-Agreement, we aimed to preserve much of the flexibility intended by the WS-Agreement authors. We designed a unique method for using domain specific predicates in the expressions. Any predicate may be used as long as it is added to the ontology and a rule is created to define the semantics of that predicate. The tool is already aware of the WSLA predicates less, lessEqual, greater, greaterEqual, equals, true, false, before. However the user is not limited to only these predicates and can define additional predicates for the domain.

3.3 Semantic Web Services

Semantic Web Services (SWS) provide an approach for representing the functionality of Web services with the help of ontologies. Popular approaches for SWS include OWL-S [13], WSMO [26], FLOWS [19] and WSDL-S [17][27]. For the the purposes of this paper, we have implemented the prototype using ontologies. The OntConcept tag annotates the SLO and Qualifying Condition parameters which facilitates the understanding and matching of the guarantee terms of the agreement. The Agreement Service Description Terms (SDT) refer to the operations of the WSDL to which the Agreement pertains. These SDT are also used during the monitoring of the service and negotiation. Both the XML based WSDLs and WS-Agreements are limited in their ability to express rich semantic meaning. In order to achieve the most accurate monitoring and negotiation the WSDL files to which the SDTs refer are semantically annotated using WSDL-S [27].

WSDL-S builds on current standards and allows multiple semantic representation languages to annotate services. This flexibility allows Web Services to be annotated with concepts from multiple ontologies from different sources. One of the most pressing challenges when mapping WSDL with ontologies is the heterogeneity between the XML Schema of the WSDL and the ontology, however, WSDL-S overcomes this challenge by providing support for rich mapping. Figure 1 shows these mappings between an Agreement and Web Service, Agreement and ontology, and Web Service and Ontology, in the context of the contract farming use case which is described in Section 5 of this paper. WS-Agreement negotiations and the runtime monitoring of WS-Agreement compliance is facilitated and enhanced by the use of semantically annotated Web Services since the ontologies provide a common understanding of the functional properties of Web Services. These semantic annotations enrich negotiations by linking heterogeneously expressed service elements to a common ontological concept. They enhance the monitoring of WS-Agreement compliance by disambiguating the terms used within the agreements and WSDL files and by providing additional domain knowledge which can be used when monitoring.

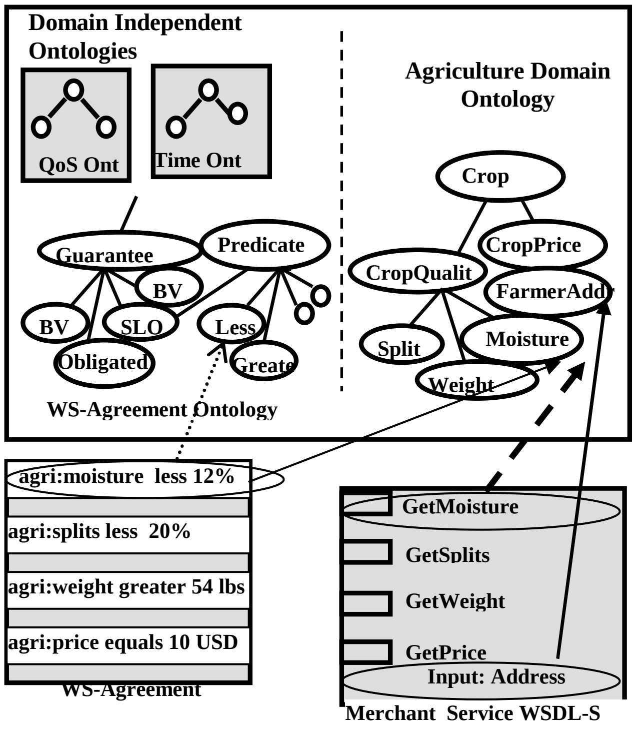


Figure 1. Illustrates the linking of Web Service and WS-Agreement concepts with ontologies.

3.4 WS-Agreement Matching

In order for a provider to be considered a suitable partner match for a given consumer, its description must contain one alternative which may satisfy any of the consumer's alternatives as denoted by the "ExactlyOne" and "ALL" tags. An agreement A contains alternative sets of Guarantee Terms such that:

A={Alt1, Alt2, …, AltN}

Alt={G1, G2, ...GN} and G={Scope, Obligated, SLO, QC, BV}

We define the following functions to facilitate the description:

"requirement(Alt, G)" returns true if G is a requirement of Alt

"capability(Alt, G)" returns true if G is an assurance of Alt

"scope(G)" returns the scope of G

"obligation(G)" returns the obligated party of G

"satisfies(Gj, Gi)" returns true if the SLO of Gj is equivalent to or stronger than the SLO of Gi

An alternative Alt1 is a suitable match for Alt2 if:

$(\forall Gi)$ such that $Gi \in Alt1 \wedge$ requirement(Alt1, Gi) $\wedge$ $(\exists Gj)$ such that $Gj \in Alt2 \wedge$ capability(Alt2, Gj) $\wedge$ scope(Gi) = scope(Gj) $\wedge$ obligation(Gi) = obligation(Gj) $\wedge$ satisfies(Gj, Gi)

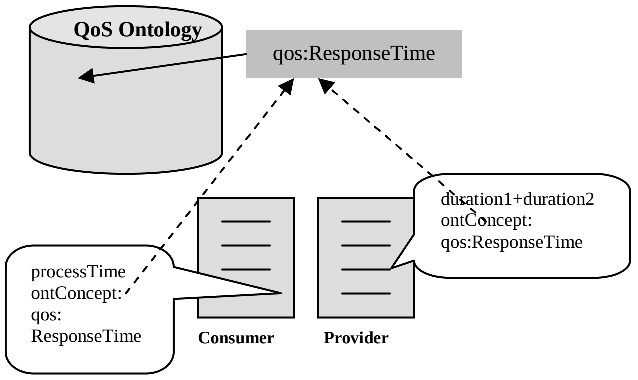

QoS Ontology
qos:ResponseTime
duration1+duration2 ontConcept: qos:ResponseTime
processTime ontConcept: qos: ResponseTime
Consumer
Provider

Figure 2. Illustrates the benefits of the ontConcept annotation.

Most users have preferences for conditions and business values and a tradeoff is decided. For instance, a user may choose an agreement with a less preferred condition but a higher penalty. Alternatively, a user with a high number of requests on the weekend would find a provider to be unsuitable if he has a condition which states that he is only able to satisfy a guarantee if it is a weekday. We consider the tradeoff between qualifying conditions and business values to be a matter of user preference and have designed a unique and flexible method for specifying these user preferences in order to yield the most suitable matches. Our approach is presented in detail in Section 4.

4. SEMANTIC WS-AGREEMENT PARTNER SELECTION

We present our framework and implementation in this section. We begin by describing the system architecture followed by how ontologies and rules were utilized to achieve better matches and to simplify the search algorithm. We then walkthrough an example which illustrates the reasoning methodology used by the tool.

4.1 Architecture

The system consists of three phases: parsing, matching and searching, which can be seen in Figure 3. To reason about domain ontologies, we use Snobase [9], an ontology based management system that offers DQL-based [5] Java API for querying OWL ontologies. IBM's ABLE engine [3] is used by Snobase for inferencing and we use ABLE Rule Language (ARL) [3] to write the rules. The ontologies are loaded into Snobase followed by each provider's WS-Agreement. We parse the agreements and load them into the system as instances of the WS-Agreement ontology. As each of these new agreement instances is created, the ABLE rule engine within Snobase executes rules as the criteria for each rule is met. The additional assertions made by the rules are used to greatly simplify the search phase by making the match decisions a priori. These rules provide additional knowledge about the domain and, as described in Section 2, play a significant role in the discovery of the most accurate match results. We discuss the rules in further detail in the next section. When a consumer seeks a partner, the consumer agreement is parsed and entered into the system as another agreement instance. The search phase begins as the algorithm considers the agreement instances and the assertions previously set by the rules and returns a list, ranked by preference, of all of the provider agreements which accurately matched the consumer's agreement.

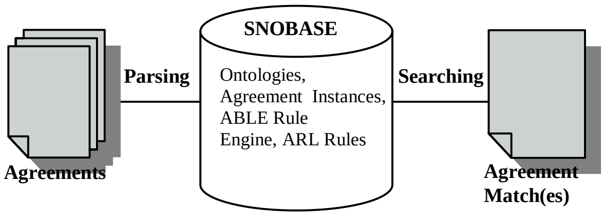

SNOBASE
Parsing
Ontologies, Agreement Instances, ABLE Rule Engine, ARL Rules
Searching
Agreements
Agreement Match(es)

Figure 3. The control flow throughout SWAPS

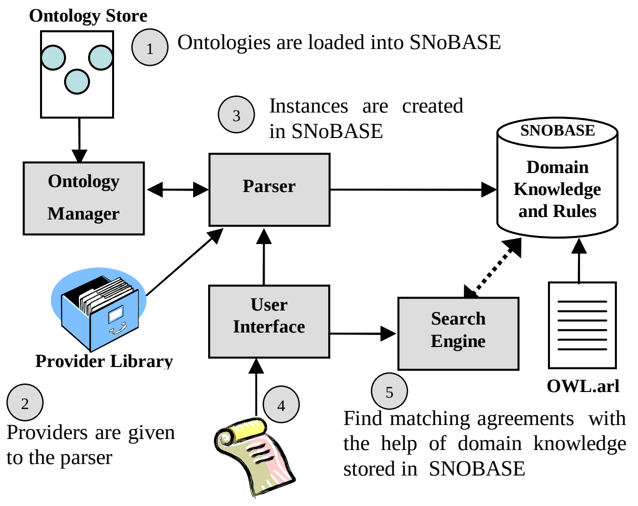
Figure 4 illustrates the system architecture. The main components of SWAPS include the ontology store, provider library, parser, ontology manager, and search engine. It is assumed that the consumer seeking a match has a library of agreement instances previously made between providers and is searching for the provider who is most able to satisfy the requirements. However it is also known that previously unknown providers, in the form of an agreement offer or a template, are constantly introduced into the set of options. Further details regarding the ontologies, rules and search engine are given in sections 4.2 and 4.3.

4.2 WS-Agreement and Rules Representation

Ontologies allow the matcher to understand the semantics of the domain; therefore enabling a much more accurate search than a syntactic approach. Rules allow for richer domain knowledge by stating additional domain rules and semantics and provide a high level of flexibility by stating customized user preferences.

4.2.1 Knowledge Representation

In order to realistically model the domains we employ several ontologies. We developed an OWL ontology to represent the WS-Agreement schema. This ontology contains the concepts from the schema such as Guarantee, Scope, and ServiceLevelObjective with relationships between them. In addition to the significant elements from the WS-Agreement, we have also included the common predicates from the WSLA specification [25]. We allow the user to add additional predicates to this ontology to preserve flexibility. An instance of this ontology is created for each agreement that is introduced into the system where they can be queried and reasoned easily. Most of the guarantees are asserted over quality of service (QoS) concepts; therefore the QoS ontology as described in [12] defines such concepts as *failureRate, latency, throughput, availability,* and *responseTime*. In addition to these ontologies a third OWL ontology represents domain specific knowledge. For our scenario in e-commerce and its implementation we are using the RosettaNet ontology (http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/rosetta.owl), also represented in OWL. Depending on the application, alternative or additional domain ontologies could be used. Finally, we use the OWL time [14] ontology to represent temporal concepts such as *endTime, interval, dayOfWeek,* and *seconds*. These ontologies are used to provide a commonality of terms between agreement parties and to provide rich domain knowledge to the search engine so that it may achieve the best possible match results.



Ontology Store
1
Ontologies are loaded into SNoBASE
3
Instances are created in SNoBASE
SNOBASE
Domain Knowledge and Rules
Ontology Manager
Parser
User Interface
Search Engine
Provider Library
OWL.arl
2
Providers are given to the parser
4
5
Find matching agreements with the help of domain knowledge stored in SNOBASE

Figure 4. SWAPS Architecture

4.2.2 Representation of Rules

We enhance the efficiency and flexibility of our matches by defining several categories of rules. These rules are represented in ARL for ABLE inferencing. The rules assert new facts if the right conditions exist for executing the various rules. We use these rules to supplement domain knowledge, convert SLOs into a common comparable form, define the semantics of domain specific predicates, and specify user preferences. Using rules instead of writing Java code to perform all of the above allows us to separate the core implementation from the user so that he may customize the matcher to the domain and personal preferences without any programming ability. We define four categories of rules and show corresponding examples below.

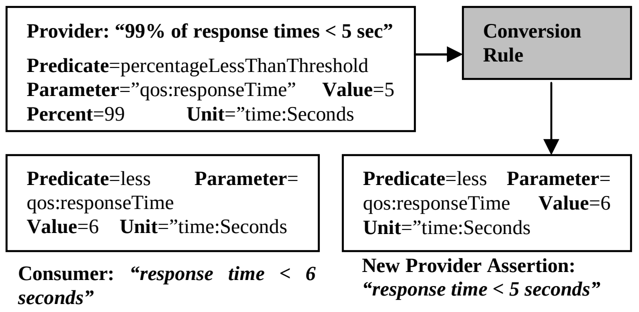
1. Conversion of Heterogeneous SLOs

Often SLOs state the same objective but express it differently. We define a category of rules to address SLOs that have semantic similarity but are syntactically heterogeneous as in the example in Figure 5. In the example, the provider is expressing an assurance using the WSLA predicate "PercentageLessThanThreshold" and the consumer is expressing the same requirement more directly using the predicate "less". While a human reader can clearly see that the provider's SLO satisfactorily meets the consumer's requirements, the heterogeneity of the predicates prevents the direct comparison of the provider and consumer SLOs. We define the following ARL rule, where *x* is a user defined threshold, to convert the provider's SLO so that it expresses the objective more directly:

when: Agreement (A) and hasGuarantee (A,G) and hasSLO (G, SLO) and hasExpression(SLO, E) and hasPredicate(E, P) and hasType(P, "PercentageLessThanThreshold") and hasPercentage(E, percent) **do:** if (percent<=*x*) then assert hasType(P, "less") else assert hasType(P, "greater")

The above ARL rule looks for any expression which contains the predicate "PercentageLessThanThreshold" and if the percentage less than *x* it changes the predicate to "less" otherwise it changes it to "greater".

In many cases the value of *x* is dependent upon the parameter. For example, a user may require a high percentage for *responseTime* but may be more lenient about other parameters. This feature can be further customized by adding additional statements in the *when* segment which perform parameter checks.



Provider: "99% of response times < 5 sec"
Predicate=percentageLessThanThreshold
Parameter="qos:responseTime" Value=5
Percent=99 Unit="time:Seconds
Conversion Rule
Predicate=less Parameter= qos:responseTime
Value=6 Unit="time:Seconds
Predicate=less Parameter= qos:responseTime Value=6
Unit="time:Seconds
Consumer: "response time < 6 seconds"
New Provider Assertion: "response time < 5 seconds"

Figure 5. Illustrates the Conversion of Heterogeneous SLOs

2. Semantics of Predicates Rules

The second category of rules allows a user to utilize any domain specific predicates within an SLO by defining how two SLOs with that predicate should be compared. A semantics rule should compare SLOs according to the predicate semantics and assert an isStronger or isEquivalent triple into Snobase. The following ARL rule defines the semantics of the predicate "less".

```
when: Agreement (A1) and hasGuaranteeTerm(A1, G1) and
hasSLObjective(G1, SLO1) and    hasExpression (SLO1, E1)
and hasPredicate(E1, P1) and hasType(P1, "less") and
hasParameter(E1, p1) and hasValue(E1, V1) and Agreement (A2)
where A1 != A2 and hasGuaranteeTerm(A2,G2) and
hasSLO(G2, SLO2) and hasExpression (SLO2, E2) and
hasPredicate(E2, P2) and hasType(P2, "less") and
hasParameter(E2, p2) and p2 == p1 and  hasValue(E2, V2)
    do:   if (V1<V2) assert [E1 isStronger E2]
            else if (V1>V2) assert [E2 isStronger E2]
            else assert [E1 isEquivalent E2]
```

The above rule compares the values of SLOs from different agreements with the same predicate and parameter and asserts isEquivalent if the values are the same otherwise it states which expression is stronger based on the semantics of the predicate "less". This rule can also be further customized by incorporating parameters or checking units to determine whether to do a string or numeric comparison. The benefit of this approach is two-fold. First, it allows for domain predicate flexibility such that we do not restrict which predicates our matcher can compare but rather allow the user to introduce new predicates by defining the semantics with an ARL rule. Second, since rules are fired automatically as the agreements are being loaded into Snobase, the SLOs are compared much before the search process. This simplifies the search algorithm because to find a match for SLO1 we quickly query for all SLOs who have been asserted isStronger than or isEquivalent to SLO1. The semantics of predicate rules have the lowest priority so that the other rules may execute before the final evaluation is performed.

3. Domain Specific Rules

The domain rules provide the matcher with richer knowledge of the domain. The following example is based on the scenario from Section 2. Consider the following domain rule for *Availability:*

MTBF is the Mean Time Between Failures

MTTR is the Mean Time To Recover

Availability = MTBF/(MTBF + MTTR)

Consider a provider agreement with the following guarantees:

Guarantee1: *SLO:* qos:MTBF=150 time:minutes, Qualifying Condition: numRequests<1000, Penalty: 5 USD, Importance 8

Guarantee2: *SLO:* qos:MTTR<5 time:minutes, Qualifying Condition: numUsers<500, Penalty: 3 USD, Importance 4

The ARL rule for *Availability* creates a new guarantee term for any agreement which has SLOs regarding both *MTBF* and *MTTR*. The new guarantee has an SLO for the A*vailability*. Any Qualifying Conditions will be compounded and a Penalty/Reward will be the higher of the two. If each has the business value importance, it will become the average of the two values. The following ARL rule accomplishes the above:

when: Agreement (A) and hasGuarantee (A, G1) and hasSLO (G1, SLO1) and hasQualifyingCondition(G1, QC1) and hasPenalty(G1, P1) and hasImportance(G1, I1) and hasExpression (SLO1, E1) and hasParameter(E1, *"qos:MTBF"*) and hasValue(E1, X) and hasGuarantee (A, G2) and hasSLO (G2, SLO2) and hasQualifyingCondition(G2, QC2) and hasPenalty(G2, P2) and hasImportance(G2, I2) and hasExpression (SLO2, E2) and hasParameter(E2, *"qos:MTTR"*) and hasValue(E2, Y) **do:** hasGuarantee (A,G3) and hasSLO(G3, SLO3) and hasExpression(SLO3, E3) and hasParameter(E3, "*qos:Availability"*) and hasVaule(E3, X+Y) and hasPenalty (G3, max(P1, P2)) and hasImportance(avg(I1,I2))

The rule will be fired once the provider agreement is loaded into Snobase and will add the following guarantee to the agreement:

Guarantee3: *SLO:* qos:Availability=96.8, Qualifying Condition: numUsers<500 AND numRequests<1000, Penalty: 5 USD, Importance: 6

4. User Preference Rules

The preference rules enable user assertions over subjective personal preferences. There is no standard of comparison for Qualifying Conditions and Business Values as they are a matter of user preference. For example, one service may be more active during the weekend in which case a provider with a condition stating that the objective may only be guaranteed if it is a weekday would not be suitable for that user. The matcher is unaware of the personal circumstances of each user until they are defined using rules. A rule may assert one of two possible assertions which will have an impact on matching: *isPreferred* or *notSuitable*. A user may write a rule to assert that "a guarantee that has a condition that the day of the week must be a weekday is not suitable" or "a guarantee with a condition involving transactionRate is preferred over a guarantee with a condition involving the day of the week". These rules have the flexibility to be more specific or generic. The following ARL rule asserts that a weekday condition is not suitable for this user:

when: Agreement (A) and hasGuarantee (A, G1) and hasQualifyingCondition(G1, QC1) which hasExpression(QC1, E1) and hasParameter(E1, "time:dayOfWeek") and hasValue(E1, "time:weekday") **do:** assert Guarantee *notSuitable G1*

The above rule asserts that a guarantee is *notSuitable* if the parameter of the Qualifying Condition is the *dayOfWeek* and if the value is *weekday*. Conflicting rules are resolved by using optional priority and condition fields.

4.3 SWAPS Search Algorithm

The system uses a two fold approach to finding the result set of providers. First, matching is automatically performed by the semantics of predicates rules as agreement instances are created. These rules significantly simplify the matching process because they compare the SLOs upon their entrance into Snobase. At this time assertions are made about which SLOs are stronger than or equivalent to other SLOs these assertions are queried by the search engine. Second, searching is done to determine which providers had agreements which were best suited for the consumer's agreement. We now detail the search algorithm. The following functions are defined to facilitate the expression of the search algorithm:

"requirement(Alt, G)" returns true if G is a requirement of Alt
"capability(Alt, G)" returns true if G is an assurance of Alt
"scope(G)" returns the scope of G
"obligation(G)" returns the obligated party of G
"isStronger(Gj, Gi)" returns true if the SLO of Gj has an assertion *isStronger* than the SLO of Gi
"isEquivalent(Gi, Gj)" returns true if the SLOs of the guarantees have the assertion *isEquivalent*
"notSuitable(G)" returns true if G has an assertion *notSuitable*

As discussed in section 2.3, matching two agreements is reduced to finding two matching alternatives and finding matching alternatives is reduced to finding matching guarantees.

$(\forall Gi)$ such that $Gi \in Alt1 \wedge Alt1 \in A1 \wedge$ requirement $(Alt1, Gi) \wedge (\exists Gj)$ S.T. $Gj \in Alt2 \wedge Alt2 \in A2 \wedge$ capability$(Alt2, Gj) \wedge$ scope(Gi)=scope$(Gj) \wedge$ obligation(Gi)=obligation$(Gj) \wedge$ (sStronger$(Gj, Gi) \vee$ isEquivalent$(Gi, Gj)) \wedge \neg$notSuitable(Gj)

4.3.1 Classification of Results

The search algorithm will yield a Vector of potential providers where each provider contains at least one alternative which can be fully satisfied and is also able to fulfill the requirements of the consumer. This set will not contain any providers which have conditions that would not be suitable for the consumer. As discussed earlier, each user will have a subjective personal preference regarding qualifying conditions and business values. If the method for stating preferences was utilized then there may be isPreferred assertions stated over some of the guarantees. We implement a preference score for each alternative which is incremented for each isPreferred statement asserted over one of the guarantees of the alternative. The agreements containing alternatives with the highest preference scores are displayed first.

4.4 Example

In this section we present an example to illustrate our approach. Table 3 shows simplified set of guarantees for a consumer. The consumer is seeking the potential providers from the library of providers given in Table 4. The tags and structure of the agreements are removed for simplicity and clarity.

Table 3. Summary of Consumer Guarantees

	Consumer1.wsag
G1	**Scope:** ProcessRequest, **Obligated:** ServiceConsumer **SLOc1:** qos:availableMemory greater 12 MB
G2	**Scope:** ProcessRequest, **Obligated:** ServiceProvider **SLOc2:** qos:failurePerWeek less 7
G3	**Scope:** ProcessRequest, **Obligated:** ServiceProvider **SLOc3:** qos:allowIncompleteInputs true
G4	**Scope:** ProcessRequest, **Obligated:** ServiceProvider **SLOc4:** 99% of qos:responseTime less 14 seconds

4.4.1 Parsing, Instance Creation and Rule Execution

When the tool is started, each of the provider agreement documents in the library given in Table 4 are parsed and loaded into Snobase. An agreement instance is created for each provider alternative. Provider 3 will have two agreement instances associated with it because it has two alternatives. As each agreement instance is loaded, the rule engine executes the rules as the criteria for each is met. The user's system includes all of the ARL rules from the previous examples in addition to a similar rule to define the semantics of "greater". An additional domain rule exists for *responseTime = processTime + transmitTime* which follows the same procedure as the previous domain rule for *Availability* but sums the values. The following rule defines the semantics of the "true" and "false" predicates:

> when any two guarantees from a different agreement instance have the same parameter and each predicate ="true" or each predicate="false"
> assert [SLO1 *isEquivalent* SLO2]

Table 5 shows the assertions as each agreement is parsed and entered into Snobase.

4.4.2 Searching

The consumer is matched against each alternative of each rovider. By querying for *isStronger and isEquivalent* assertions for the Provider's SLOs, the algorithm determines that Provider 1 is able to satisfy the consumer's needs and the consumer can also satisfy the requirement expressed in G1. However, Provider 1 is dismissed as a potential match because one of the guarantees was asserted as *notSuitable* as highlighted in number 3 of Table 5.

Provider 2's first alternative is considered and the algorithm will determine that not all of the consumer's guarantees are satisfied as the provider does not have an isStronger or isEquivalent assertion for each of them and as one of the SLOs is weaker than the consumer SLO as highlighted in number 9 from Table 5. The algorithm moves on to the next alternative of Provider 2 and determines that it is a match because all of the consumer's guarantees are satisfied and none of the relevant provider guarantees have been asserted as *notSuitable*. The algorithm returns Provider 2 as the only match.

4.4.3 Post Search Considerations

There was only one potential match in the simplified example above. However, if there had been more compatible providers in the library, the algorithm would continue with additional steps. There are several issues of preference in the example above. If Provider1 had been a suitable match the *responseTime* is guaranteed to be less than 14 seconds with a very high penalty of 15 USD. Provider 2 offers a much faster *responseTime* of 9 seconds but a much lower penalty of 1 USD. Some users may desire efficiency while others may wish to merely satisfy the objective while sacrificing some efficiency for the potential of a high penalty payoff. Since this is a personal user preference, the user may define a rule which states that a guarantee *isPreferred* if the penalty is over some threshold. The user may also wish to state that if the penalties are the same then faster speeds are preferred. During the matching process, the preference score for each alternative is incremented each time a satisfactory guarantee has the *isPreferred* assertion. When multiple providers are able to satisfy the basic needs of a consumer, the results are ranked by highest preference scores so that the user may consider the most preferred providers first. This example showed the reasoning

Table 4. Summary of Guarantees from Provider Library

	Provider1.wsag	**Provider2.wsag (Provider2a and Provider2b)**		
G1	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO1:** qos:responseTime less 14 sec. **QC:** time:dayOfWeek equals weekday **Penalty:** 15 USD	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO5:** qos:transmitTime less 4 sec. **QC:**qos:maxNumUsers less 1000 **Penalty:** 1 USD	**OR**	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO9:** qos:transmitTime less 4 sec. **QC:** qos:maxNumUsers less 1000 **Penalty:** 1 USD
G2	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO2:** qos:failurePerWeek less 7 **Penalty:** 10 USD	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO6:** qos:processTime less 5 sec. **QC:** qos:numRequests less 500 **Penalty:** 1 USD		**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO10:** qos:processTime less 5 sec. **QC:** qos:numRequests less 500 **Penalty:** 1 USD
G3	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO3:** qos:incompleteInputs true	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO7:** qos:failurePerWeek less 16 **Penalty:** 2 USD		**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO11:** qos:failurePerWeek less 7 **Penalty:** 2 USD
G4	**Scope:** ProcessRequest **Obligated:** ServiceConsumer **SLO4:** qos:availableMemory greater 12MB	**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO8:** qos:incompleteInputs false		**Scope:** ProcessRequest **Obligated:** ServiceProvider **SLO12:** qos:incompleteInputs true

process while illustrating the flexibility provided by the user defined rules.

Table 5. SWAPS Matching

	Guarantee	Fact/Rule	Assertion
1	Consumer G4	PercentageLessThan Threshold Conversion Rule	qos:responseTime < 14 seconds
2	Provider1 G1	Semantics of "less"	SLO1 *isEquivalent* SLOc4
3	Provider1 G1	User Preference Rule weekday *notSuitable*	Provider1's G4 *notSuitable*
4	Provider1 G2	Semantics of "less"	SLO2 *isEquivalent* SLOc2
5	Provider1 G3	Semantics of "true"	SLO3 *isEquivalent* SLOc3
6	Provider1 G4	Semantics of "greater"	SLOc1 *isStronger* SLO4
7	Provider2a G1 and G2	Domain rule for "qos:ResponseTime"	Provider2a-*G5-SLO13:* qos:responseTime less 9 secs., Qualifying Condition:numRequests< 1000 AND numUsers<500 Penalty: 1 USD
8	Provider2a G5	Semantics of "less"	SLO13 *isStronger* SLOc4
9	Provider2a G3	Semantics of "less"	SLOc2 *isStronger* SLO7
10	Provider2b G1 and G2	Domain rule for "qos:ResponseTime"	Provider2b-*G5-SLO14:* qos:responseTime less 9 secs., Qualifying Condition:numRequests< 1000 AND numUsers<500 Penalty: 1 USD
11	Provider2b G5	Semantics of "less"	SLO14 *isStronger* SLOc4
12	Provider2b G3	Semantics of "less"	SLO11 *isEquivalent* SLOc2
13	Provider2b G4	Semantics of "true"	SLO12 *isEquivalent* SLOc3

5. APPLICATION OF AGREEMENTS AND AGREEMENT MATCHING

This section does not attempt to show another technical example but rather describes how WS-Agreements and our tool can be applied to remedy a challenging real world situation. The next sections will describe the problem, how WS-Agreements can be applied, and how the WS-Agreement matching tool can solve this problem.

5.1 Agriculture in India

Agricultural trade in India is problematic for both Farmers and Merchants and there is a lack of effective use of IT to facilitate trade. Farmers spend time and resources growing goods and sending them to the markets without guarantee that they will be sold. The farmer pays for the transportation of the goods and the wastages that occur when the goods spoil during transport. Merchants have no assurances on the quality or availability of the goods that they seek to purchase. This problem is addressed in [4] and the authors describe an Agricultural Information System to improve the effectiveness of decision-making in the agriculture domain. A Web Services based business process management system developed to aid the marketing of agricultural produce is described in [18]. Each party involved is represented as a Web Service. If each party is a Web Service, then the process of matching farmer to merchant can be reduced to one of Web Service composition and policy matching.

Contract farming is one remedy currently being practiced to solve the dilemma and is described in [6] as a system for the production and supply of agricultural products under forward contracts between producers/suppliers and buyers. The cultivator makes a commitment to provide an agricultural commodity of a certain type, at a time and a price, and in the quantity required by a known and committed buyer. Using faming contracts, growers and buyers can agree to terms and conditions for the sale and purchase of goods. The buyer can make agreements to supply selected goods which sometimes also include land preparation and technical advice. The contracts ensure that the grower follows recommended production methods and cultivation and harvesting specifications. Conditions are frequently stated regarding the price and quality of goods and penalties in the form of discounts are offered for flaws or lack of quality.

The situation for farmers is improved as they no longer must send goods to markets without a guarantee of acquisition. The farmer's price risk is reduced because the contracts specify the prices in advance. The buyers obtain more consistent quality and more reliable production than if purchases were made on the open market. When efficiently organized and managed, contract farming reduces risk and uncertainty for both parties as compared to buying and selling crops on the open market. The success stories of E-Chaupal and Tata Kisan Sansar, who have implemented contract farming in India, are discussed in [18]. Just as Web Services can represent the farmers and merchants, the WS-Agreement is well suited to represent the complex contracts drawn between the two.

5.2 Contracts as WS-Agreements

The WS-Agreement is perhaps the best suited standard for representing farming contracts. The protocol is functional for representing the guarantees which always include some objective and often contain conditions which must exist in order for the objective to be fulfilled. For example, a merchant may guarantee a price under the condition that the goods are of a certain quality. Business values such as penalties are often seen in contracts in the form of discounts. For example, a farmer may guarantee that the moisture percentage is than 10% and may offer a discount for every bushel that contradicts that assurance. In this case, a merchant is considered to be a service consumer and his guarantees and requirements can be proficiently represented using WS-Agreement. The available merchants are the service providers. Table 6 contains an example of farming contracts as WS-Agreements. It depicts the merchant as the consumer seeking the most suitable farmer, however, this tool can also be used by a farmer to find the ideal merchant. Section 5.3 will discuss how SWAPS can easily match a merchant with a farmer who will provide the required quality at a desired price.

5.3 WS-Agreement Matching for the Agriculture Domain

An ontology representing the Agriculture domain can provide the matcher with a complete understanding of the domain and the user can supplement this knowledge with rules specific to the domain. The user can also write any relevant conversion rules for measurements. For example, the user may write a rule to convert from ounces to grams or from bushels to pounds. For predicates, this user may which to use the basic predicates already defined within the system or can also add domain specific predicates. The simple example in Table 6 uses predefined predicates. In this domain, price is compared differently than moisture or splits because, with the latter, both parties specify that the number must be less than some value because while moisture may vary per bushel it must always be less than some value. Price, however, is a fixed price per bushel. Therefore, when comparing price, expressions with different predicates may still be compatible. For example, the merchant is willing to pay five cents or less but the farmer is asking 4 cents or greater per bushel. Since a parameter such as price will be reasoned over differently than a parameter like moisture, a separate rule must be defined to define the procedure for comparing price. The user will surely have personal preferences and may define these as rules. In Table 6, Farmer 1 clearly offers better quality goods while Farmer 2 offers much higher penalties. The merchant may specify the tradeoff as an ARL rule which states that high penalties are preferred. This causes Farmer 2 to be presented as a higher match than Farmer 1. This tool can effectively narrow down the hundreds of farmers into a group which contains only those farmers offering what the merchant requires. The merchant can specify additional preferences and aspects which are *notSuitable* to further narrow down the search. Finally the merchant is presented with one or more farmers, in order of preference, from which to choose. This feature greatly reduces the search effort for both farmers and merchants. It can ensure that each farmer and merchant gets the best possible deal tailored to their individual needs and preferences.

Table 6. Farming Contracts represented with WS- Agreement

Merchant	Farmer 1	Farmer 2
Guarantee1: SLO1: Moisture is less inclusive 12% **Guarantee2: SLO2:** splits is less inclusive 20% **Guarantee3: SLO3:** test weight is greater than 54 lbs **Guarantee4: SLO4:** price lessEqual 10 cents per bushel	**Guarantee1: SLO1:** Moisture is less 10% **Penalty:** discount $10 each **Guarantee2: SLO2:** splits is less inclusive 20% **Penalty:** splits of 5% or more, discount $1 each **Guarantee3: SLO3:** test weight is greater than 60 lbs **Guarantee4: SLO4:** price greaterEqual 8	**Guarantee1: SLO1:** Moisture is less inclusive 12% **Penalty:** discount $15 each **Guarantee2: SLO2:** splits is less inclusive 20% **Penalty:** splits of 3% or more, discount $5 each **Guarantee3: SLO3:** test weight is greater than 58 lbs **Guarantee4: SLO4:** price greaterEqual 7

6. RELATED WORK

There has been very little work done in the area of WS-Agreement. A formal definition of the WS-Agreement is given in [1] and the schema is extended by adding tags to accommodate states for negotiation. Cremona [11] is a tool for the creation and monitoring of WS-Agreements. Both contributions do not consider partnering agreements. Major work in the domain of Service Level Agreement (SLA) matching is purely syntactic. [28] developed a methodology for matching Web Service Level Agreements (WSLA). This work syntactically matches SLAs by parsing them into syntax trees. The authors have designed a matching algorithm which compares these trees node by node. Heterogeneous SLAs are handled by referencing a table containing instructions which the code must execute in order to convert them into the same format. Such syntactic approaches must take a more exhaustive and laborious approach to matchmaking and are challenged by less obvious matches. Since our agreements are parsed into instances of the WS-Agreement OWL ontology, we are able to reason over the ontology and retrieve data via ontology queries with much less effort. In addition, the semantics defined by this ontology result in more accurate matches. This work focuses on matching Service Level Objectives, where, our work considers compatible scopes and SLOs to be the most essential criteria for matching but also reason over qualifying conditions and business values. GlueQoS [29] extends the grammar of WS-Policy to add qualifying conditions. This approach uses only XML based models which limits the expressivity of the assertions. Since XML cannot express formal meaning, the matching is purely syntactic which greatly limits efficiency of the matching process. Our work uses the combination of OWL ontologies and ARL rules to provide our matcher with detailed knowledge of the domain, QoS, and agreements which leads to better matches.

The following work uses rules without semantics to represent policies. Paschke et al use a rule based SLA language (RBSLA) to express Service Level Agreements in [16]. RBSLA is an extension of RuleML tailored to satisfy the requirements of the SLA domain. The rules are based on the logic components of Derivation, Event Condition, Event Calculus, Courteous Logic, Deontic Logic, and Description Logic. Rule based SLAs can be written and modified using the management tool (RBSLM) which also enables the management, maintenance and monitoring of contract rules. Policy matching is not considered in the scope of this approach. There has also been some work that has benefited from using Semantic Web technologies. Uszok et al have developed KAOS for the specification, management, analysis, and enforcement of policies [20]. The policy is represented using concepts from an OWL ontology. Role-value maps are added to later work to compensate for some of the limitations in expressiveness of OWL. The trust and privacy of Web services is handled with a rule based engine in [7], and in [8], the authors discuss the combination of OWL ontologies and SWRL rules. Parsia et al present the OWL ontology developed for representing policies however they do not utilize rules [15]. Li et al apply a very interesting approach to Access Control Policy specification [10]. Access Control Policies are designed and expressed using a combination of OWL and SWRL. Policies are defined using an ontology. SWRL is introduced to enhance OWL with additional expressiveness and deducible ability. Access control policies are designed in the form of rules using concepts defined in the ontology and relationships such as *isPermittedDoWith* to express which kinds of agents have permission to access resources. This work aims to express policies and does not consider the matching of these policies. Verma et al presents a successful approach to policy matching by combining semantics with rules to achieve efficient matches. WS-Policy is extended to incorporate semantics and policies are represented using an OWL ontology. SWRL rules express additional domain concepts and expand the matching ability. Our work applies a similar approach to WS-Agreement and extends it to also reason over scope, qualifying conditions and business values. We provide matching flexibility by allowing users to define their own predicates and preferences.

7. CONCLUSION AND FUTURE WORK

This work presents a novel contribution to the area of WS-Agreement and agreement matching. With the framework and implementation described throughout this paper, service providers and consumers may automatically make the most accurate and effective partnerships which are tailored to user preferences. While this objective has been considered in the prior works, we extend this by defining reasoning methods for the Scopes, Obligations, SLOs, Qualifying Conditions, and Business Values of the Guarantee Terms. We consider the subjectivity of the latter two and implement a feature which allows for the specification of what the user prefers and what the user considers unsuitable. We effectively match complex agreements containing multiple alternatives and symmetry such that both consumer and provider have capabilities *and* requirements. This work utilizes an effective combination of ARL rules with multiple Ontologies in order to achieve flexibility and accuracy. In the process it demonstrates the need and value of annotating multiple activities (e-commerce in our exanple) with non-functional and domain-independent ontologies. Use of WSDL-S for semantic Web Services is also demonstrated in this context. We define several categories of rules to enhance domain specific knowledge, efficiently handle heterogeneous SLOs, allow the definition of user preferences, and flexibly allow domain specific predicates while greatly simplifying the matching process. These rules are a powerful addition because they allow the matching process to be changed and customized at any time without any modifications to the code or programming knowledge.

Since a key feature of our work is to customize the matching process with user defined rules, this work will benefit from a module which converts rules defined with SWRL to ARL rules to facilitate the definition of rules by user. This tool can be extended to incorporate negotiations as defined by the protocol in [2]. Suitable agreements can be identified by the current tool and negotiations between parties could ensue. This tool can also be augmented to support other standards for policy specification such as WS-Policy. This would allow consumer to provider matches regardless of the specification used. This kind of matchmaking can be integrated with the METEOR-S configuration and runtime binding middleware [22].

Acknowledgements: We thank Dr. Heiko Ludwig (IBM), a member of WS-Agreement committee, for his guidance. Prof. Sanjay Chaudhari of DA-IICT, India outlined the agriculture scenario. We also acknowledge partial support from the IBM Eclipse Innovation Grant and the collaboration with the METEOR-S project team, of which this research is a component.

8. REFERENCES

[1] Aiello, M., Frankova, G., and Malfatti, D. What's in an Agreement? An Analysis and an Extension of WS-Agreement, Proc. 3rd *ICSOC, 2005*

[2] Andrieux, A., Czajkowski, C., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M., *WebServices Agreement Specification (WS-Agreement).* June 29th 2005

[3] Bigus, J.P., Schlosnagle, D.A., Pilgrim, J.R, Mills III, W.N., and Diao, Y. ABLE: A toolkit for building multiagent autonomic systems, IBM Systems Journal, 41 (3), 2002

[4] Chaudhary, S., Sorathia, V., Laliwala, Z., Architecture of Sensor based Agricultural Information System for Effective Planning of Farm Activities. IEEE SCC 2004: 93-100

[5] DQL Technical Committee 2003. DAML Query Language (DQL). http://www.daml.org/dql

[6] Eaton, C., Shepherd, A., Contract Farming Partnerships for Growth FAO Agricultural Services Bulletin 145

[7] Kagal, L., Paoucci, M., Srinivasan, N., Denker, G., Finin, T., and Sycara, K. Authorization and Privacy for Semantic Web Services, AAAI Spring Symposium on SW S, 2004

[8] Kagal, L., Finin, T., and Joshi, A. Declarative Policies for Describing Web Service Capabilities and Constraints, Proceedings of W3C Workshop on Constraints and Capabilities for Web Services, 2005

[9] Lee, J., Goodwin, R. T., Akkiraju, R., Doshi, P., Ye, Y. Snobase: A Semantic Network-based Ontology Ontology Management http://alphaWorks.ibm.com/tech/Snobase 2003

[10] Li, H., Zhang, X., Wu, H., Yuzhong, Q., Design and Application of Rule Based Access Control Policies. Proc of the Semantic Web and Policy Workshop, 2005, Galway, IR.

[11] Ludwig, H., Dan, A., Kearney, B., Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. Proc 2nd *ICSOC*,, New York, 2004.

[12] Maxemilien, M., Singh, M., A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing 8(5):84-93, September-October 2004

[13] OWL-S, http://www.daml.org/services/owl-s/

[14] Pan, F., Hobbs, J. OWL Time http://www.isi.edu/~pan/damltime/time-entry.owl

[15] Parsia, B., Kolovski, V., Hendler, J. Expressing WS-Policies in OWL. Policy Management for the Web Wkshp, May 2005

[16] Paschke, A., Dietrich, J., Kuhla, K. A Logic Based SLA Management Framework. Proc. of the Semantic Web and Policy Workshop, November, 2005.

[17] Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., Adding Semantics to Web Services Standards, ICWS 2003

[18] Sorathia, V., Laliwala, Z., and Chaudhary, S. Towards Agricultural Marketing Reforms: Web Services Orchestration Approach, IEEE SCC 2005.

[19] SWSF, http://www.w3.org/Submission/SWSF/

[20] Uszok, A., Bradshaw, J.M., Jeffers, R., Johnson, M., Tate, A., Dalton, J., Aitken, S. Policy and Contract Management for Semantic Web Services, Proc. of the AAAI Spring Symposium on Semantic Web Services, 2004

[21] Verma, K., Akkiraju, R., Goodwin, R. Semantic Matching of Web Service Policies, SDWP Workshop, 2005.,

[22] Verma K., Gomadam K., Lathem, J., Sheth A., Miller, J., Semantically Enabled Dynamic Process Configuration, LSDIS Lab Technical Report 2006.

[23] The Web Service Policy Framework, http://www-106.ibm.com/developerworkds/library/ws-polfram

[24] Wohlstadter, E., Tai, S., Mikalsen, T., Rouvello, I., Devanbu, P. GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions, The Proc ICSE 2004, pp. 189-199

[25] The WSLA Specification, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[26] Web Services Modeling Ontology, http://www.wsmo.org

[27] WSDL-S, http://www.w3.org/Submission/WSDL-S/

[28] W. Yang, H. Ludwig, A. Dan: Compatibility Analysis of WSLA Service Level Objectives. Workshop on the Design of Self-Managing Systems. Supplemental, 2003