Extending the SweetDeal Approach for E-Procurement using SweetRules and RuleML

Sumit Bhansali and Benjamin N. Grosof

Massachusetts Institute of Technology,
Sloan School of Management, Cambridge, MA 02139, USA
{bhansali,bgrosof}@mit.edu; http://ebusiness.mit.edu/bgrosof

Abstract. We show the first detailed realistic e-business application scenario that uses and exploits capabilities of the SweetRules V2.1 toolset for e-contracting using the SweetDeal approach. SweetRules is a uniquely powerful integrated set of tools for semantic web rules and ontologies. SweetDeal is a rule-based approach to representation of business contracts that enables software agents to create, evaluate, negotiate and execute contacts with substantial automation and modularity. The scenario that we implement is of electronic procurement of computers, with request-response iterated B2B supply-chain management communications using RuleML as content of the contracting discovery/negotiation messages. In particular, the capabilities newly exploited include: SweetJess or SweetXSB to do inferencing in addition to the option of SweetCR inferencing, SweetOnto to incorporate/merge-in OWL-DLP ontologies, and effectors to launch real-world actions. We identify desirable additional aspects of query and message management to incorporate into RuleML and give the design of experimental extensions to the RuleML schema/model, motivated by those, that include specifically: fact queries and answers to them. We present first scenario of using SCLP RuleML for rebates and financing options, in particular exploiting the courteous prioritized conflict handling feature. We give a new SweetDeal architecture for the business messaging aspect of contracting, in particular exploiting the situated feature to exchange rulesets, that obviates the need to write new (non-rule-based) agents as in the previous SweetDeal V1 prototype. We finally analyze how the above techniques, and SweetDeal, RuleML and SweetRules overall, can combine powerfully with other e-business technologies such as RosettaNet and ebXML.

1 Introduction

In this paper, we describe in detail a practical electronic contracting scenario that uses RuleML[1], the Situated Courteous Logic Programs (SCLP) knowledge representation [6], and the SweetRules V2.1 semantic web rules toolset [2] together to show how a real-world business application such as electronic procurement can be supported with semantic web technologies including also OWL [3]. The electronic procurement application was chosen not only because of its wide applicability in e-business but also because it allows us to showcase different features of the new SweetRules V2 implementation. Specifically, we show how powerful features of the new implementation such as importing OWL-DLP ontologies into a rule-based knowledge base, executing real-world business processes such as sending e-mail from rules, and inferencing on RuleML rules obtained from ontologies as well as rulebases possibly expressed in different types of KR. The procurement example allows us to also see how different business functions/features such as rebates, financing scenarios, payment options, which might be applicable in a wide variety of business applications, can be expressed using the RuleML KR language.

From our investigation of the electronic procurement scenario, we suggest inclusion of specific features in future versions of the RuleML KR to support query and message management that would be useful especially in business applications involving iterated request-response communication, such as e-contracting applications. Finally, we also explain how our electronic contracting approach based on RuleML and SweetRules can relate to other e-business technologies such as RosettaNet [4] and ebXML [5].

The paper is organized as follows. In section 2, we provide a brief overview of the technologies – RuleML, SweetRules, and SweetDeal – that we use in this research. Section 3 provides an overview of our approach and scenario. Section 4 illustrates the expressive power of RuleML in representing key contract provisions, specifically those of financial incentives. Section 5 describes the iterated contract construction process in great detail. Section 6 concludes the paper.

2 Overview of Technologies

We provide below a short description of the different technologies used in this research.

2.1 RuleML

RuleML [1] is the emerging standard for representing semantic web rules. The fundamental KR used in RuleML is situated courteous logic program or SCLP, which has been demonstrated to be expressively powerful [6]. The courteous part of SCLP enables prioritized conflict handling, which in turn enables modularity in specification, modification, merging and updating. The situated part of SCLP enables attached procedures for "sensing" (i.e. testing rule antecedents) and "effecting" (i.e. performing actions when certain conclusions are reached).

2.2 SweetRules

SweetRules [2], a uniquely powerful integrated set of tools for semantic web rules and ontologies, is newly enhanced in V2.1. The new version of SweetRules include capabilities such as first-of-a-kind semantics-preserving translation and interoperability between a variety of rule and ontology languages (including XSB Prolog [7], Jess [8] production rules, HP Jena-2 [9], IBM CommonRules [10], and the SWRL [11] subset of RuleML), highly scaleable backward and forward inferencing, and easy merging of heterogeneous distributed rulebases/ontologies.

2.3 SweetDeal

SweetDeal [12] is an electronic contracting approach that uses SCLP RuleML to support creation, evaluation, negotiation, execution and monitoring of formal electronic contracts between agents such as buyers and sellers. The approach builds on top of the SweetRules toolest to showcase the power of SCLP, RuleML, and SweetRules, as a design -- and implemented prototype software -- in the specific business application of electronic contracting.

3 Overview of Approach and Scenario

The extended SweetDeal approach described in this paper consists of three primary pieces: communication protocol between the contracting agents, contract knowledge bases and agent communication knowledge bases. We briefly describe these below in the context of our specific scenario of electronic procurement.

3.1 Communication Protocol

In our scenario, the buyer, Acme Corp, is interested in purchasing computers of a particular configuration. The buyer attempts to establish a procurement contract with the seller, Dell Computers. We assume that Dell Computers is a preferred vendor of computers for Acme Corp. To establish the terms of the contract, the buyer and seller agents exchange messages in an iterated fashion.

The protocol of message exchanges is as follows: the buyer first sends an RFP (request for proposal) to the seller. The seller responds to the RFP with the proposal. Based on specific business criteria, the buyer chooses to accept or reject the proposal. The buyer may also suggest modifications to the proposal before

accepting or rejecting it. The RFP message from the buyer contains specific details about the desired computer configuration. It also contains any queries to which the seller must provide answers in its proposal. The proposal message from the seller contains several formal contract fragments which describe useful business provisions such as rebates, financing options, as well as payment options for the buyer. In addition to specifying the contractual provisions, the seller also provides answers to the queries posed by the buyer. Finally, it may pose additional queries for the buyer that the buyer in turn must provide answers to in the next negotiation message. After the buyer is satisfied with the final contract proposal from the seller, it generates a purchase order that is sent to the seller. To complete the transaction, the seller delivers the order and the buyer makes arrangements to pay the seller via the chosen payment option. Any contingencies in the execution of the order/transaction are handled according to the terms of the contract.

3.2 Contract Knowledge Bases

Contract negotiation messages exchanged between the agents are RuleML knowledge bases that are executable within SweetRules V2.1 software. Contract knowledge bases contain the following six main technical components: rules, facts, ontologies including OWL-based ontologies as well as object-oriented default inheritance ontologies, effectors, f-queries and their answers, and conditional queries. We briefly describe each of these components below. Since RuleML as an XML-based markup language is fairly verbose and since the presentation syntax of RuleML has not yet been implemented completely in SweetRules, we use the IBM CommonRules (CR) V3.3 syntax in all our examples to allow for concise presentation and easier comprehension. In future, it would be more desirable instead to use the RuleML presentation syntax. See [16], especially the Rules language description, for the initial version of that presentation syntax, and see [2], especially its documentation, for its experimental extension to include the Situated feature and for its (currently, still partial) support in SweetRules.

3.2.1 Rules

RuleML rules express the if-then implications of the contractual fragments and form the bulk of the contract knowledge base. Each rule has a head and a body. The “head” is the part of the rule after the “then”, whereas the “body” is the part of the rule that follows “if” and precedes “then”. The example below shows a simple <rebate> rule: the seller might wish to provide a rebate offer to the buyer in the proposal. Specifically, the seller might wish to offer a rebate in the amount of $1000 to the buyer if the number of computers ordered by the buyer is more than 75. Due to current tool limitations of numeric types in translating CommonRules to RuleML, all numeric constants in the rule examples below are represented using strings, e.g., “75” is represented as “seventyfive”.

```
<rebate>
if
       quoteID(?QuoteID) AND quantityOfItemOrdered(?Q) AND
              isGreaterThan(?Q, seventyfive)
then
       rebateAmount(?QuoteID, thousand);
```

3.2.2 Facts

RuleML facts or assertions are rules that have no bodies. The simple examples below show facts that are specified in the RFP from the buyer to the seller. The quantity of item ordered by the buyer is 80 (computers) and the buyer is located in the state of Florida. (We assume that both buyer and seller are located in USA).

```
       quantityOfItemOrdered(eighty);
       buyerLocationState(florida);
```

3.2.3 Ontologies

Ontologies are vocabularies that express the background knowledge used by the contract rules. They can be either OWL [15] ontologies or rule-based object-oriented default inheritance ontologies. OWL ontologies used must be in the Description Logic Programs (DLP) [13] subset of OWL, i.e. in the subset of OWL that is translatable into LP rules. SweetRules V2.1 software allows for translation from OWL-DLP to RuleML rules. We show below a simple example of an OWL ontology that is used by the buyer. The ontology (procurement.owl) has three classes: *buyer*, *seller*, and *product*, and three object properties: *preferredVendorIs*, *buysProduct*, and *sellsProduct*. The ontology fragment also has some instance data: *computers is a product, Dell sells computers, Acme buys computers, Acme has Dell as a preferred vendor*. Since the ontology is in the DLP subset of OWL, a translation from OWL to RuleML exists and SweetRules V2.1 software can be used (see command C1 below) to convert the ontology to a rule-based knowledge base in RuleML.

```
translate owl clp c:\procurement.owl c:\procurement.clp          (C1)
```

The ontology (procurement.owl) is shown below:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.procurement.org/procurement.owl#"
    xml:base="http://www.procurement.org/procurement.owl">
  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="buyer"/>
  <owl:Class rdf:ID="seller"/>
  <owl:Class rdf:ID="product"/>

  <owl:ObjectProperty rdf:ID="preferredVendorIs">
        <rdfs:domain rdf:resource="#buyer"/>
        <rdfs:range rdf:resource="#seller"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="buysProduct">
        <rdfs:domain rdf:resource="#buyer"/>
        <rdfs:range rdf:resource="#product"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="sellsProduct">
        <rdfs:domain rdf:resource="#seller"/>
        <rdfs:range rdf:resource="#product"/>
  </owl:ObjectProperty>

  <seller rdf:ID="dell">
       <sellsProduct rdf:resource="#computers"/>
  </seller>

  <buyer rdf:ID="acme">
        <preferredVendorIs rdf:resource="#dell"/>
         <buysProduct rdf:resource="#computers"/>
  </buyer>

  <product rdf:ID="computers"/>
</rdf:RDF>
```

The translation of the ontology to rules is shown below. The translation has been slightly modified for ease of readability. Each of the predicates below would be prefixed in the original translation with a long

namespace URI indicated in the OWL document above. The namespace URI has been removed from all predicates below.

```
<emptyLabel> if buysProduct(?X, ?Y) then buyer(?X);
<emptyLabel> if buysProduct(?X, ?Y) then product(?Y);
<emptyLabel> if sellsProduct(?X, ?Y) then seller(?X);
<emptyLabel> if sellsProduct(?X, ?Y) then product(?Y);
<emptyLabel> if preferredVendorIs(?X,?Y) then buyer(?X);
<emptyLabel> if preferredVendorIs(?X, ?Y) then seller(?Y);
<emptyLabel> sellsProduct(dell, computers);
<emptyLabel> preferredVendorIs(acme, dell);
<emptyLabel> buyer(acme);
<emptyLabel> product(computers);
<emptyLabel> Class(product);
<emptyLabel> Class(buyer);
<emptyLabel> Class(seller);
<emptyLabel> seller(dell);
<emptyLabel> buysProduct(acme, computers);
```

Next we show a simple example of expressing an object-oriented default inheritance ontology using rules. In the example, *BuyWithCredit* is a subclass of *Buy*. *Buy* assigns the value "invoice" to the *paymentMode* property, but *BuyWithCredit* assigns the value "credit" to the *paymentMode* property, i.e., *BuyWithCredit* overrides the *paymentMode* property inherited by default from *Buy*. The courteous feature of SCLP RuleML is a powerful way to express default inheritance using rules. If only Buy(quoteID) is asserted (i.e. the buyer asserts that it wants to buy), then the payment mode is assumed to be invoice (by default). If the buyer specifically asserts BuyWithCredit(quoteID), then the default payment mode is overridden to be credit instead.

```
<buyRegular>if Buy(?quoteID) then paymentMode(?quoteID,invoice);
/* BuyWithCredit is a subclass of Buy */
if BuyWithCredit(?quoteID) then Buy(?quoteID);
<buyCredit> if BuyWithCredit(?quoteID)then paymentMode(?quoteID,credit);
overrides(buyCredit, buyRegular);
```

3.2.4 Effectors

Effectors are a feature of the Situated extension of logic programs. An effector procedure is an attached procedure that is associated with a particular predicate. This association is specified via an effector statement that is part of the rulebase. When a conclusion is drawn about the predicate, an action is triggered; this action is the invocation of the effector procedure, and is side-effect-ful. In general, there may be multiple such effector statements and procedures in a given rulebase, e.g.., in a given SweetDeal contract/proposal. Effectors can execute real-world business processes associated with the execution of the contract. For example, an effector can be used by the buyer to send the purchase order (PO) to the seller (see <sendPO> rule below). If the vendor proposal has been approved, then the buyer sends the PO to the sales e-mail address of the vendor. The effector *sendPOtoVendor* is associated with the Java procedure *emailMessage* in the *Effector_EmailPO* class, whose path is indicated as *com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs*.
The Java procedure not shown here for brevity handles the e-mail messaging aspect of sending the PO to the vendor. The arguments to the effector predicate – seller e-mail address, location of the purchase order, approved proposal identifier – are passed as arguments to the Java procedure.

```
<sendPO>
if
  approvedVendorProposal(?Vendor, ?ProposalID) AND
  emailSalesAddress(?Vendor, ?SellerAddress) AND locationOfPO(?Location)
then
  sendPOtoVendor(?SellerAddress, ?Location, ?ProposalID);
```

```
<emptylabel>
   Effector: sendPOtoVendor
   Class: Effector_EmailPO
   Method: emailMessage
   path:
"com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs";
```

3.2.5 Fact-queries or F-queries

The traditional notion of the answer to a query in logic programs (and databases) is: a set of variable-binding lists. In modeling the exchange of contract proposals and associated dialogue between contracting parties, however, it is often convenient to model the answer to an inquiry as a set of facts instead. Accordingly, we have developed the design of f-queries (short for "fact queries") as a (fairly simple) experimental extension to RuleML. Note that, unlike the rest of what we describe of the SweetDeal approach in this paper, this f-queries feature is *not yet implemented in SweetRules*. RuleML f-queries are queries which have facts as their answers. They facilitate the iterated development of procurement contracts. The example below shows a sample f-query. It is an f-query from buyer to seller in which the buyer requests the seller for the unitPriceOfItem. The answer to the f-query is provided by the seller as a RuleML fact.

```
Query Example
<query>
  <_body>
    <fclit cneg="no" fneg="no">
      <_opr>
        <rel>unitPriceOfItem</rel>
      </_opr>
      <var>QuoteID</var>
      <var>Price</var>
    </fclit>
  </_body>
</query>
```

3.3 Agent Communication Knowledge Bases

In addition to the contract knowledge bases that are shared/exchanged, the agents also have internal RuleML knowledge bases that contain rules to facilitate agent communication. The effectors feature of SCLP RuleML allows the agents to execute real-world business processes such as e-mail messaging. This feature is used by the agents to send the contract rulesets to each other. The actual e-mail messaging effector procedure is implemented as a Java method that employs the JavaMail API [14]. The communication process is triggered using the internal agent communication KB and the SweetRules V2.1 software that supports execution of Java methods attached as effectors to specified predicates in the KB. A simple example follows: the situated rule <sendRFP> allows the buyer to send the RFP ruleset to the sales e-mail address of the seller. The name of the effector in the situated rule is sendRFPtoComputerSeller. The effector specification consists of the name of the Java procedure (emailMessage), the Java implementation class that contains the method (Effector_EmailRFP), and the path to the class (com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs).

The effector is executed when the buyer wants to buy computers and the seller sells computers and is in the preferred vendor list of the buyer. When the sendRFPtoComputerSeller predicate is concluded, the attached procedure "emailMessage" is called to execute the required action. The action consists of reading the RFP from the local file system and sending it via e-mail to the specified e-mail address of the sales department of the seller. For brevity, the Java code to implement the e-mail messaging is not shown here.

```
<sendRFP>
if
   wantToBuy(?Buyer, computers) AND seller(?Vendor) AND
```

```
       sell(?Vendor, computers) AND inPreferredVendorList(?Buyer, ?Vendor) AND
       emailSalesAddress(?Vendor, ?Address) AND
       locationofRFP(?Buyer, computers, ?Location)
then
   sendRFPtoComputerSeller(?Address, ?Location);

<emptylabel>
   Effector: sendRFPtoComputerSeller
   Class: Effector_EmailRFP
   Method: emailMessage
   path:
"com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs";
```

4 Contract Business Provisions using RuleML

In this section, we present a few key contract fragments in the procurement contracting scenario and how SCLP RuleML can be used to express them. We intend to show how the expressive/declarative power of RuleML allows for easy addition and modification of key B2B contracting provisions. Specifically, we focus on expressing commonly used financial incentives such as rebates, discount pricing, and financing options. These incentives could be specified by the seller in its proposal. For the sake of simplicity and brevity, in this paper version some of the rules (e.g., about monthly payments in financing options) are highly specific to the particular scenario, rather than specified in more realistically general form.

4.1 Rebate

For example: the seller wishes to offer a rebate in the amount of $1000 to the buyer if the quantity of item ordered is greater than 75. This is represented as the <rebate> rule below.

```
<rebate>
if
       quoteID(?QuoteID) AND quantityOfItemOrdered(?Q) AND
       isGreaterThan(?Q, seventyfive)
then
       rebateAmount(?QuoteID, thousand);
```

4.2 Pricing Options

For example: If the buyer makes the purchase before April 1 then the unit price offered by the seller is $600; if the purchase is made before April 15, then the unit price offered is $650. This is specified as the <earlyPurchase> and <latePurchase> rules below. If both these rules apply, i.e., if the purchase was made before April 1, then precedence is given to the earlyPurchase rule. This precedence is specified using the courteous prioritization feature of SCLP (and of RuleML): see the overrides fact rule below.

```
<earlyPurchase>
if
       quoteID(?QuoteID) AND purchaseDate(?QuoteID, ?Date) AND
       isLessThan(?Date, oneApr05)
then
       unitPriceOfItem(?QuoteID, sixhundred);

<latePurchase>
if
       quoteID(?QuoteID) AND purchaseDate(?QuoteID, ?Date) AND
       isLessThan(?Date, fifteenApr05)
then
       unitPriceOfItem(?QuoteID, sixhundredfifty);

overrides(earlyPurchase, latePurchase);
```

```
MUTEX
       unitPriceOfItem(?QuoteID, sixhundred) AND
       unitPriceOfItem(?QuoteID, sixhundredfifty);
```

4.3 Financing Option

For example: If the financing is requested for 36 months by the buyer, the unit price of the item is determined to be $600, and the quantity ordered is 50, then the financing option offered by the seller is such that the monthly payment is $958 and the total interest paid is $4500 (see the <financing> rule below).

```
<financing>
if
       quoteID(?QuoteID) AND financeForMonths(?QuoteID, thirtysixMonths) AND
       unitPriceOfItem(?QuoteID, sixhundred) AND
       quantityOfItemOrdered(?QuoteID, fifty)
then
       monthlyPayment(?QuoteID, ninehundredfiftyeight) AND
       totalInterest(?QuoteID, fourthousandfivehundred);
```

5 Details of Procurement Contract Construction Using RuleML and SweetRules V2.1

In this section, we describe in detail the specific steps taken in constructing an e-contract between the buyer and seller using SCLP RuleML and SweetRules V2.1 in our electronic procurement scenario.

As described earlier, the buyer has a solo (or unshared) agent communication knowledge base that can be used to initiate the action of sending an RFP to a specific seller (in our example – Dell). We call this solo knowledge base – BSO1. BSO1 has the names of the different sellers, types of products offered by them, their respective sales e-mail addresses, and whether the sellers are in the preferred vendor list maintained by the buyer. The location of the RFP (which itself is a rule-based knowledge base) is indicated using the *locationofRFP* predicate. The rule that triggers sending the RFP to the seller is indicated by <sendRFP>: if the buyer wants to buy computers and the seller sells computers and is in the preferred vendor list of the buyer, send the RFP from the indicated local filesystem location to the seller's sales e-mail address. The predicate *sendRFPtoComputerSeller* is associated with the situated effector procedure *emailMessage*, which uses the JavaMail API to send the RFP ruleset to the seller via e-mail.

Buyer Solo KB – BSO1

```
wantToBuy(acme, computers);
seller(dell);
seller(staples);
sell(dell, computers);
sell(staples, officesupplies);
inPreferredVendorList(acme, dell);
inPreferredVendorList(acme, staples);
emailSalesAddress(dell, "sales@dell.com");
emailSalesAddress(staples, "sales@staples.com");
locationofRFP(acme, computers, "c:\\buyertosellerRFP.clp");

<sendRFP>
if
   wantToBuy(?Buyer, computers) AND seller(?Vendor) AND
  sell(?Vendor, computers) AND inPreferredVendorList(?Buyer, ?Vendor) AND
  emailSalesAddress(?Vendor, ?Address) AND
  locationofRFP(?Buyer, computers, ?Location)
then
  sendRFPtoComputerSeller(?Address, ?Location);
```

```
<emptylabel>
   Effector: sendRFPtoComputerSeller
   Class: Effector_EmailRFP
   Method: emailMessage
   path:
"com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs";
```

In SweetRules V2.1, the "exhaustForwardInfer" command is given to derive all the conclusions from a given rulebase, and along with those conclusions to perform all the associated effecting actions that those conclusions trigger (i.e., sanction). For example, the command C2 below generates all the conclusions of BSO1 and (as an effecting action) sends the RFP to the seller. The "clp" in the first two arguments of the command indicates that CommonRules V3.3. format is the input and output knowledge base format, the third argument gives the location of BSO1, and the fourth argument specifies that IBM CommonRules should be used indirectly as an underlying inference engine when performing inferencing. SweetRules V2.1 software allows for a choice of such underlying engines. In our example, SweetRules enables Jess or XSB, as well as CommonRules, to be used as indirect underlying engine; for each choice of underlying engine, it would generate semantically equivalent conclusions and perform the same set of triggered effecting actions

```
exhaustForwardInfer clp clp c:\buyertosellerSendRFP.clp CommonRules      (C2)
```

The RFP sent by the buyer to the seller is a collection of rules. The RFP consists of two parts -- a shared knowledge base that contains most importantly the required computer configuration details (we call this knowledge base BSH1) and a set of f-queries that request specific answers from the seller (we call this set of queries BFQ1).

BSH1 indicates the buyer name, quantity of item ordered, buyer state, and the required computer configuration details. The rule <checkOfferedConfiguration> is used by the buyer to check whether the vendor offered configuration satisfies the minimum requirements. Since RuleML built-ins are not currently directly and smoothly supported in SweetRules V2.1 beyond the SWRL subset of RuleML, we also provide several facts to support arithmetic comparison.

Buyer to Seller RFP (BSH1)

```
buyerName(acme);  /* buyer name is acme */
quantityOfItemOrdered(fifty);  /* quantity of item ordered is fifty */

/* buyer is located in the state of Florida */
buyerLocationState(florida);

/* speed of processor should be at least 2GHz */
requiredMinProcessorSpeedInGHZ(twogigahertz);
if
 requiredMinProcessorSpeedInGHZ(?Speed) and
 offeredProcessorSpeedInGHZ(?OfferSpeed) and isGreaterThan(?OfferSpeed, ?Speed)
then
 isSpeedAcceptable(true);

/* not shown here for brevity:  there are also additional computer system
configuration details (memory size, hard disk storage capacity, monitor size,
monitor type (flat?), monitor resolution) */ ...
...

/* check if the configuration is acceptable */
<checkOfferedConfiguration>
if
  isSpeedAcceptable(true) and isMemorySizeAcceptable(true) and
  isHardDiskSizeAcceptable(true) and isMonitorSizeAcceptable(true) and
  offeredMonitorType(flat) and
  offeredMonitorResolution(tenTwentyFourBySevenSixtyEight)
then
  isOfferedConfigurationAcceptable(true);
```

```
/* The following are some facts in lieu of arithmetic built-ins. */
isGreaterThan(fourgigahertz, twogigahertz);
isGreaterThan(onezerotwofourmb, fivetwelvemb);
isGreaterThan(sixtyGB, fortyGB);
isGreaterThan(seventeen, fifteen);
```

BFQ1 is the collection of f-queries that ask the seller to specify the vendor quote identifier, the offered computer configuration details, the unit price of item, taxes as percent of price, service charge as percent of price, delivery charges for shipment, and the delivery time in days. For brevity, only a few of the f-queries are shown below.

Buyer to Seller f-Queries (BFQ1)

```
<rulebase>
  <_rbaselab>
    <ind>FQueries</ind>
  </_rbaselab>
  <query>
    <_body>
      <fclit cneg="no" fneg="no">
        <_opr>
          <rel>quoteID</rel>
        </_opr>
       <var>QuoteID</var>
      </fclit>
    </_body>
  </query>
  <query>
    <_body>
      <fclit cneg="no" fneg="no">
        <_opr>
          <rel>offeredProcessorSpeedInGHZ</rel>
        </_opr>
       <var>Speed</var>
      </fclit>
    </_body>
  </query>
...
```

After the seller receives the RFP, the seller sends its rule-based contract proposal to the buyer. The proposal contains three parts – BSH1 (i.e. shared knowledge base transmitted from buyer to seller – see above), answers to f-queries posed by the buyer plus the shared knowledge base that contains rules about pricing, rebates, financing options and other business provisions (we call this SSH1), and lastly f-queries for the buyer (SFQ1).

Seller to Buyer (SSH1)

```
/* quote ID is 1 */
quoteID(one);
/* computer configuration details */
offeredProcessorSpeedInGHZ(fourgigahertz);
offeredSizeofmemoryInMB(onezerotwofourmb);
offeredSizeofharddiskInGB(sixtyGB);
offeredMonitorSizeInInches(seventeen);
offeredMonitorType(flat);
offeredMonitorResolution(tenTwentyFourBySevenSixtyEight);

/* Pricing Rules */
/* if purchase date is before April 1 2005, then unit Price is $600;
   if purchase date is before April 15 2005, then unit Price is $650*/
<earlyPurchase>
if
```

```
       quoteID(?QuoteID) and purchaseDate(?QuoteID, ?Date) and
       isLessThan(?Date, oneApr05)
then
       unitPriceOfItem(?QuoteID, sixhundred);
<latePurchase>
if
       quoteID(?QuoteID) and purchaseDate(?QuoteID, ?Date) and
       isLessThan(?Date, fifteenApr05)
then
       unitPriceOfItem(?QuoteID, sixhundredfifty);

overrides(earlyPurchase, latePurchase);

MUTEX
       unitPriceOfItem(?QuoteID, sixhundred) and
       unitPriceOfItem(?QuoteID, sixhundredfifty);

/* there is no service charge */
if
       quoteID(?QuoteID)
then
       serviceChargeAsPercentOfPrice(?QuoteID, zeroPercent);

/* Delivery Options */

/* if delivery type is standard then delivery charge is $2500 for the order
   if delivery type is express then delivery charge is $5000 for the order
*/
<standard>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, standard)
then
       deliveryChargesForShipment(?QuoteID, twentyfivehundred);

<express>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, express)
then
       deliveryChargesForShipment(?QuoteID, fivethousand);
MUTEX
       deliveryType(?QuoteID, standard) and deliveryType(?QuoteID, express);

/* if delivery type is standard then delivery time in days is 14 days
   if delivery type is express then delivery time in days is 7 days
*/
<standardDeliveryTime>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, standard)
then
       deliveryTimeInDays(?QuoteID, fourteendays);

<expressDeliveryTime>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, express)
then
       deliveryTimeInDays(?QuoteID, sevendays);

MUTEX
       deliveryTimeInDays(?QuoteID, fourteendays) and
deliveryTimeInDays(?QuoteID, sevendays);

/* Additional assertions from Seller */
/* Financial Incentives section */
```

```
/* not shown here for brevity: the financial incentives of discount pricing,
rebate, financing option already shown above in section 4 */
...
/* Sales Tax */
/* no sales tax in Florida */
<tax0>
if
       quoteID(?QuoteID) and buyerLocationState(florida)
then
       taxesAsPercent(?QuoteID, zeroPercent);

/* 5% sales tax in states other than Florida */
<tax5>
if
       quoteID(?QuoteID) and buyerLocationState(?X) and NotEquals(?X, florida)
then
       taxesAsPercent(?QuoteID, fivePercent);

MUTEX
       taxesAsPercent(?QuoteID, zeroPercent) and
       taxesAsPercent(?QuoteID, fivePercent);

/* Object-oriented default inheritance using rules */
/* If you buy, then default payment mode is invoice */
<buyRegular> if Buy(?QuoteID) then
paymentMode(?QuoteID, invoice);

/* BuyWithCredit is a subclass of Buy */
if BuyWithCredit(?QuoteID) then Buy(?QuoteID);

<buyCredit>
if BuyWithCredit(?QuoteID) then paymentMode(?QuoteID, credit);

overrides(buyCredit, buyRegular);

MUTEX
       paymentMode(?QuoteID, credit) and paymentMode(?QuoteID, invoice);

isLessThan(twentyfiveMarch05, oneApr05);
isLessThan(twentyfiveMarch05, fifteenApr05);
isLessThan(fiveApr05, fifteenApr05);
isGreaterThan(eighty, seventyfive);
NotEquals(massachusetts, florida);
```

SFQ1 is a collection of f-queries posed by the seller for the buyer. The seller asks whether the buyer would like to buy and whether the buyer would like to buy with a credit card. The seller also queries for the purchase date, delivery type and number of months of financing requested. For brevity, only a few of the f-queries are shown below.

Seller to Buyer F-Queries (SFQ1)

```
<rulebase>
  <_rbaselab>
  <ind>FQueries</ind>
  </_rbaselab>
  <query>
    <_body>
       <fclit cneg="no" fneg="no">
         <_opr>
              <rel>purchaseDate</rel>
          </_opr>
         <var>QuoteID</var>
         <var>Date</var>
```

```
        </fclit>
    </_body>
  </query>
...
```

When the buyer receives the proposal ruleset from the seller, it answers the queries posed by the seller (see BA1 below) and then performs exhaustive inferencing on the resulting ruleset (BSH1 + SSH1 + BA1) to obtain the derived conclusion set (CS1). Logical inferencing allows the buyer to determine the key parameters (such as unit price, delivery charges, taxes, etc.) of the proposal and also whether the proposal meets minimum specified criteria in the RFP.

Answers to F-Queries posed by seller (BA1)

```
Buy(one);
BuyWithCredit(one);
purchaseDate(one, fiveApr05);
deliveryType(one, express);
financeForMonths(one, thirtysixMonths);
```

The conclusion set (CS1) tells the buyer that the offered configuration is acceptable, unit price of item will be $650, delivery time will be 7 days, % discount already included in the price is 13%, taxes are 5%, rebate amount is $1000, and payment mode is credit.

Conclusion Set (CS1) obtained from BSH1 + SSH1 + BA1

```
isLessThan(twentyfiveMarch05, oneApr05);
isLessThan(twentyfiveMarch05, fifteenApr05);
isLessThan(fiveApr05, fifteenApr05);
requiredMinProcessorSpeedInGHZ(twogigahertz);
quoteID(one);
requiredMinSizeofmemoryInMB(fivetwelvemb);
offeredSizeofmemoryInMB(onezerotwofourmb);
requiredMonitorResoluton(tenTwentyFourBySevenSixtyEight);
purchaseDate(one, fiveApr05);
quantityOfItemOrdered(eighty);
BuyWithCredit(one);
deliveryType(one, express);
NotEquals(massachusetts, florida);
isGreaterThan(fourgigahertz, twogigahertz);
isGreaterThan(onezerotwofourmb, fivetwelvemb);
isGreaterThan(sixtyGB, fortyGB);
isGreaterThan(seventeen, fifteen);
isGreaterThan(eighty, seventyfive);
creditCardNumber(one, ccNumber9876543298765432);
offeredSizeofharddiskInGB(sixtyGB);
overrides(earlyPurchase, latePurchase);
overrides(earlyPurchaseDiscount, latePurchaseDiscount);
overrides(buyCredit, buyRegular);
offeredMonitorSizeInInches(seventeen);
requiredMinSizeofharddiskInGB(fortyGB);
offeredProcessorSpeedInGHZ(fourgigahertz);
financeForMonths(one, thirtysixMonths);
requiredMonitorType(flat);
offeredMonitorType(flat);
buyerName(acme);
buyerLocationState(massachusetts);
requiredMinMonitorSizeInInches(fifteen);
offeredMonitorResolution(tenTwentyFourBySevenSixtyEight);
vendorName(dell);
serviceChargeAsPercentOfPrice(one, zeroPercent);
deliveryChargesForShipment(one, fivethousand);
isSpeedAcceptable(true);
Buy(one);
```

```
rebateAmount(one, thousand);
isMonitorSizeAcceptable(true);
isMemorySizeAcceptable(true);
isHardDiskSizeAcceptable(true);
isOfferedConfigurationAcceptable(true);
deliveryTimeInDays(one, sevendays);
discountPercentAlreadyIncluded(one, thirteen);
unitPriceOfItem(one, sixhundredfifty);
taxesAsPercent(one, fivePercent);
paymentMode(one, credit);
```

6 Relationship of other B2B Technologies to our Approach

RosettaNet and ebXML are two very important and influential approaches to XML-based e-business messaging including about contracting and e-commerce. It is desirable to be able to use our SweetDeal approach together with such XML-based e-business messaging infrastructure. In this section, we discuss how SweetDeal and (SCLP) RuleML can be used with RosettaNet and with ebXML. The punchline is that they play well together; the SweetDeal contract rulesets can be carried as the "letters" content within the "envelopes" of RosettaNet or ebXML messages, i.e., within their messaging interfaces and protocols. In doing so, it is both possible and useful to utilize the (non-OWL) ontologies provided by RosettaNet and ebXML, and to perform sending of messages as actions.

6.1 RosettaNet

Next, we begin with RosettaNet, and discuss specifically how RosettaNet Partner Interface Processes (PIPs) can be used with RuleML in the context of our electronic procurement scenario. RosettaNet is a consortium of information technology, electronic components, semiconductor manufacturing and solutions providers, which seeks to establish a common language and standard processes for business-to-business (B2B) transactions. RosettaNet PIPs define business processes between trading partners. The PIP specifies the roles of the trading partners that participate in the business process as well as the business activities that compose the process. The PIP also specifies XML-based action messages or business documents that are exchanged between the roles during business activities. The specification of a standard structure for the business documents is a major part of the PIP specification. An example of a RosettaNet PIP is PIP3A1 which provides a detailed XML message guideline for implementing the Request Quote business process. A message fragment from PIP3A1 is shown below –

```
<ContactInformation>
     <contactName>
            <FreeFormText>A</FreeFormText>
            <EmailAddress>abc@xyz.com</EmailAddress>
            …..
     </contactName>
</ContactInformation>
```

The message fragment above specifies the structure for contact information for the buyer who sends the request for quote to the seller. Our SweetDeal approach can be used straightforwardly in combination with the exchange of RosettaNet PIP messages between the two parties. We can also directly use the standardized (non-OWL) ontological terms from the PIP messages in our rulebases. For example, the request for proposal (RFP) sent by the buyer to the seller in our scenario allows for use of the ontological terms in the RosettaNet PIP3A1 XML message guidelines. A SweetDeal quote (contract proposal) rulebase cf. our earlier scenario can then employ as predicates (i.e., as ontological terms) various properties drawn from the PIP specification, e.g., the unit price of the product, which is specified in RosettaNet using the following DTD segment –

```
<!ELEMENT unitPrice ( ProductPricing ) >
<!ELEMENT ProductPricing ( FinancialAmount , GlobalPriceTypeCode ) >
<!ELEMENT FinancialAmount ( GlobalCurrencyCode , MonetaryAmount ) >
<!ELEMENT GlobalCurrencyCode ( #PCDATA ) >
<!ELEMENT MonetaryAmount ( #PCDATA ) >
```

For example, the seller would specify the following fact rule in the proposal to the buyer:

```
unitPrice(?GlobalCurrencyCode, ?MonetaryAmount).
```

6.2 ebXML

Likewise, ebXML can be used in our scenario along with RuleML and the SweetDeal approach to support electronic contracting between two parties. Both the buyer and the seller in our scenario would maintain ebXML collaboration protocol profiles (CPPs) that would describe the specific business collaborations supported by each of the parties using the ebXML business process specification schema (BPSS). For example, the buyer CPP would show that the "request for proposal" is a business process that is supported by it. The details of the "request for proposal" business process would be specified using the ebXML BPSS. The parties that will engage in the interaction protocol will reach agreement on how to collaborate by exchanging the CPPs to construct a collaboration protocol agreement (CPA), which fixes the protocol for interaction between the parties. Once agreement has been reached, ebXML messages in accordance with the collaboration agreement can be exchanged using ebMS (or ebXML Message Service). The payload of these messages can contain the RuleML rulebases to establish the electronic procurement contract.

7 Conclusions

In this paper, we have extended the SweetDeal approach and applied the extended approach using the new SweetRules V2.1 semantic web rules prototype software to a practical, real-world B2B application in the domain of electronic contracting. The electronic procurement contracting scenario that we have described in detail shows how semantic web rules technology, specifically RuleML and SweetRules, can be powerfully used in e-contracting.

Acknowledgements: Thanks to Shashidhara Ganjugunte, Chitravanu Neogy, Said Tabet, and the rest of the SweetRules development team, for helping to realize the implementation, and for useful discussions.

References

[1] Rule Markup Language Initiative, http://www.ruleml.org and http://www.mit.edu/~bgrosof/#RuleML
[2] SweetRules, http://sweetrules.projects.semwebcentral.org/
[3] OWL and the Semantic Web Activity of the World Wide Web Consortium. http://www.w3.org/2001/sw
[4] RosettaNet, http://www.rosettanet.org
[5] ebXML (ebusiness XML) standards effort, http://www.ebxml.org
[6] Grosof, B.N., "Representing E-Business Rules for Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML". Proc. Wksh. on Information Technology and Systems (WITS '01), 2001.
[7] XSB logic programming system. http://xsb.sourceforge.net/
[8] Jess (Java Expert System Shell). http://herzberg.ca.sandia.gov/jess/
[9] Jena, http://jena.sourceforge.net/
[10] IBM CommonRules. http://www.alphaworks.ibm.com and http://www.research.ibm.com/rules/
[11] SWRL, A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/
[12] Grosof, B.N., C.T. Poon, "SweetDeal: Representing Agent Contracts With Exceptions using Semantic Web Rules, Ontologies, and Process Descriptions". International Journal of Electronic Commerce (IJEC), 8(4):61-98, Summer 2004, Special Issue on Web E-Commerce.
[13] Grosof, B.N., Horrocks, I., Volz, R., and Decker, S., "Description Logic Programs: Combining Logic Programs with Description Logic". Proc. 12th Intl. Conf. on the World Wide Web (WWW-2003).
[14] JavaMail, http://java.sun.com/products/javamail/
[15] OWL Web Ontology Language, http://www.w3.org/TR/owl-features/
[16] Semantic Web Services Framework Version 1.0, http://www.daml.org/services/swsf/1.0