# A Contract Agreement Policy-based Workflow Methodology for Agents Interacting in the Semantic Web

Kalliopi Kravari[1], Grammati-Eirini Kastori[1],
Nick Bassiliades[1] and Guido Governatori[2]

[1]Dept. of Informatics, Aristotle University of Thessaloniki, GR-54124 Thessaloniki, Greece
[2]NICTA, Queensland Research Laboratory, Australia
[1]{kkravari, gkastori, nbassili}@csd.auth.gr
[2]guido.governatori@nicta.com.au

**Abstract.** The Semantic Web aims at automating Web content understanding and user request satisfaction. Intelligent agents assist towards this by performing complex actions on behalf of their users into real-life applications, such as e-Contracts, which make transactions simple by modeling the processes involved. This paper, presents a policy-based workflow methodology for efficient contract agreement among agents interacting in the Semantic Web. In addition, we present the integration of this methodology into a multi-agent knowledge-based framework, providing flexibility, reusability and interoperability of behavior between agents. The main advantage of our approach is that it provides a safe, generic, and reusable framework for modeling and monitoring e-Contract agreements, which could be used for different types of on-line transactions among agents. Furthermore, our framework is based on Semantic Web and FIPA standards, to maximize interoperability and reusability. Finally, an e-Commerce contract negotiation scenario is presented that illustrates the usability of the approach.

**Keywords:** semantic web, intelligent agents, e-Contracts, defeasible reasoning.

## 1 Introduction

The *Semantic Web* (*SW*) [1] is a rapidly evolving extension of the WWW, where the semantics of information and services is well-defined, making it possible for people and machines to understand Web content. Its penetration has transformed the way people satisfy their requests, letting them save time and money. Moreover, SW technologies offer interoperability and, thus, favor *Intelligent Agents* (*IAs*) [2]. Hence, the integration of *multi-agent systems* (*MAS*) with SW technology affects the use of the Web; groups of intercommunicating agents are available to traverse the Web and perform complex actions on behalf of their users in real-life applications. One such application is *Electronic Contracts* (*e-Contracts*), which make transactions simple by effectively modelling and managing the processes involved.

M. Dean et al (eds)
RuleML 2010, The 4th International Web Rule Symposium
LNCS 6403, pp. 223-237

In essence, a contract is an agreement that creates and modifies legal relationshiops (obligations, permissions, prohibitions) between two or more parties and involves several stages, such as information exchange and negotiation. An e-Contract, on the other hand, is a contract modelled, specified, executed and enacted (controlled and monitored) by a software system [3]. The main differentiation is that e-Contracts are simply carried out electronically, overcoming the delays and drawbacks of the manual process [4]. Thus, like ordinary contracts, e-Contracts define a set of clauses that must be satisfied by the parties involved.

The execution of an e-Contract involves, as mentioned, several tasks, which can be represented as *workflows* [5]. More specifically, workflow information is extracted from a contract and then the various functions that realize assorted activities within the specified time-bounds are implemented [6]. A workflow is concerned with the automation of procedures, where information is passed between participants according to an overall goal. Thus, the workflow of an e-Contract must be carefully specified and related to meet the contract requirements.

This paper focuses on e-Contracts managed by IAs. Each agent has its own policy, a set of private rules representing its requirements, obligations and restrictions, depending on its role in the e-Contract, as well as its personal data. The e-Contract, on the other hand, has a set of clauses that specify among others how it will be implemented and outlines restrictions on the parties involved. Taking the above into account, a policy-based workflow methodology is proposed that specifies the overall negotiation stages of an e-Contract. The aim of the methodology is to propose a safe, reusable procedure for e-Contract agreements, which could be used for different types of on-line transactions among agents.

Despite the usual issues a contract has to deal with, e-Contracts have to deal with additional issues regarding agents. As agents not necessarily share the same logic or rule representation formalism, this paper presents the integration of the above methodology into a multi-agent knowledge-based framework, called EMERALD. This framework deals with the aforementioned issues proposing the use of trusted, third party reasoning services that can be used in safely exchanging policies with heterogeneous rule formalisms. The provided advantages are, among others, flexibility (each agent can use its own rule formalism; workflow rule sets can vary from empty clauses to large and complex rule programs, etc), reusability (workflow clause sets can be reused in different scenarios and can be shared among agents, since they are modular) and interoperability (agents can use different rule formalisms which can be safely interchanged through external trusted reasoning services) of behavior between agents. Finally, an e-Commerce contract negotiation scenario is presented that illustrates the usability of the approach.

The rest of the paper is organized as follows. In Section 2, we present a policy-based workflow methodology for efficient contract agreement among agents. Section 3 presents the integration of this methodology into the multi-agent Knowledge-based framework. In Section 4, an e-Commerce contract negotiation scenario is presented that illustrates the usability of the approach. Section 5 discusses related work, and Section 6 concludes with final remarks and directions for future work.

## 2 Policy-based Workflow Methodology

After briefly presenting e-Contracts and the utility of agents in the setting, this section presents the proposed policy-based workflow methodology for contract agreement among agents.

### 2.1 E-Contracts

E-Contracts, as already mentioned, are agreements between two or more parties to create legal obligations between them, which are modelled, specified and executed by a software system [3]. They consist of at least two parties, here agents, and a number of clauses. Typically, an e-Contract is described by the abstract specification: $EC \equiv \{P, C\}$, where $P$ is the set of parties involved $P \equiv \{P_1, P_2, ..., P_n\}$, $n \geq 2$ and $C$ is the set of clauses $C \equiv \{C_1, C_2, ..., C_m\}$. Each party possesses a well-defined role, specified in the e-Contract. For example, in an on-line e-commerce transaction the parties involved may be defined either as buyers or as sellers. In this study, we assume that there are two agents $P \equiv \{P_1, P_2\}$ that want to make a contract agreement; agent $P_2$ sets out the agreement rules and agent $P_1$ negotiates over these rules.

Generally speaking, an agreement between parties is legally valid if it satisfies the requirements of the e-Contract. This intention is proven by their compliance with the clauses of the e-Contract, which, can actually be divided into stages of information exchange and negotiation. More specifically, an e-Contract can be divided into groups of tasks forming special categories; for example, each e-Contract has to contain a stage, in which the involved parties negotiate the terms of agreement, by means of an offer and acceptance that the e-Contract refers to. This natural categorization in stages is considered very useful for all parties involved, as they can better understand the e-Contract steps and, thus, organize their policy accordingly.

### 2.2 Intelligent Agents

Agents involved in an e-Contract actually act on behalf of their users, thus, they have to contract an agreement efficiently and without human intervention. In order to achieve this, each agent possesses arguments that describe its requirements, preferences and restrictions. These arguments usually include data and rules that comprise the agent's policy and characterize its behavior. A careful consideration would reveal that these policies can, like e-Contracts, be divided into groups of rules, such as personal data restriction rules. Thus, taking advantage of this analogy could lead to an automation of e-Contract procedures.

However, the variety in representation and reasoning technologies is one of the main issues in agent interoperability. An IA (Intelligent Agent) does not necessarily have to oblige to other agents' logic, nor is it essential for the agents to understand each other's rule representation format. In fact, intercommunicating agents usually "understand" different (rule) languages. Thus, it will be essential not only to come up

with an automation methodology for e-Contract procedures, but also to provide the suitable framework that will overcome the above issues in real-life applications.

### 2.3 E-Contract Agreement Workflow Automation Methodology

E-Contracts reduce costs, save time, speed up customer response and improve service quality by reducing paperwork, thus increasing automation. As mentioned, the implementation of an e-Contract involves several groups of tasks, which can be represented as workflows. However, the workflow of an e-Contract must be carefully specified, in order to meet the contract requirements. Thus, taking into account that both e-Contract clauses and participants' policies can be divided in stages, this paper proposes a policy-based workflow methodology, which divides the overall process of an e-Contract agreement in stages.

More specifically, we propose the specification of an e-Contract to be extended to an 8-tuple $EC \equiv \{P, C, N_{STG}, STG, N_{STP}, STP, C_{STP}, COND_{STP}\}$, where $N_{STG}$ is the number of stages of the policy workflow, which in our case is five, $STG$ is the set of stages $STG \equiv \{stg_l, 1 \le l \le N_{STG}\}$, $N_{STP}$ is the set of the number of steps for each stage $N_{STP} \equiv \{N^l_{STP}, 1 \le l \le N_{STG}\}$, $STP$ is the set of steps for each stage $STP \equiv \{stp_k^l, 1 \le k \le N^l_{STP}: stg_l \in STG\}$, $C_{STP}$ is the set of contract clauses for each step $C_{STP} \equiv \{C_k^l: stp_k^l \in STP \wedge C_k^l \subseteq C \wedge (\forall l,k,l',k' \; C_k^l \cap C_{k'}^{l'} = \varnothing \vee (k=k' \wedge l=l'))\}$, and $COND_{STP}$ is the set of step transition conditions $COND_{STP} \equiv \{cnd_k^l: stp_k^l \in STP\}$, which decide if the transaction can proceed from one step to the next and it is part of the agent's internal policy.

Notice that the difference between contract clauses and transition conditions is that the former refer to conditions of the contract that are publicly known, whereas the latter refer to conditions of the state of the workflow that are private to the contracting agents. In general, transitions between stages and steps are sequential, but the parties involved can disagree at any step, terminating the negotiation without agreement. Thus, in order to end in a state of agreement and eventually execute the e-Contract, each stage and step has to be successful, i.e. the set of clauses and conditions of each step should be satisfied.

Our methodology involves at each step: (a) the exchange of the agent's $P_i$ clauses ${}_iC_k^l$ to the agent $P_j$, (b) the evaluation of the ${}_iC_k^l$ clauses using agent's $P_j$ personal data ${}_jD_k^l$, and (c) the exchange of the results/conclusions ${}_jE_k^l = \mathrm{I}({}_iC_k^l \cup {}_jD_k^l)$ of step (b) from agent $P_j$ back to agent $P_i$, in order to test if the clauses of the contract are satisfied so that the contract negotiation workflow can continue. The workflow transition decision is taken by the following algorithm:

**if** ${}_jE_k^l \neq \varnothing$ **and** $\mathrm{I}({}_icnd_k^l \cup {}_jE_k^l) \neq \varnothing$
**then** (**if** $k=N^l_{STP}$ **then** $l \leftarrow l+1$; $k \leftarrow 1$ **else** $k \leftarrow k+1$)
**else** $l \leftarrow$ **T** (Termination)

### 2.4 e-Contract Agreement Workflow Steps

With the automation of e-Contract agreements, e-commerce is expected to improve productivity and competitiveness by providing unprecedented access to an on-line

global market place with millions of customers and thousands of products and services. On the other hand, since the e-Contract proposal focuses on an automated environment and not on humans, who take decisions on specific transactions, it is extremely important to avoid any fraud and discrepancy in the contract.

Thus, the first stage refers to *trust* ($stg_1$). The aim of this stage is to assure that all involved parties are trusted. However, establishing trust is mainly pertained to a problem of authorization and access control [7] [8]. In order to deal with this issue, we propose a policy-based approach, based on a set of policies and credentials (digital certificates). Usually, credentials are sufficient when the agent is convinced either of the other agent's identity or his membership in a sufficiently trusted group. Thus, in $stg_1$, the agents involved should exchange the appropriate credentials that will enable them to trust each other.

As soon as a satisfying level of trust among agents is established, the procedure can advance to the next stage ($stg_2$). This stage includes the set of steps that involve the primary data exchange that is required in an e-Contract, in order to specify its context. Thus, in this stage the agents involved could exchange, among others, definition and interpretation data, commencement and completion data or even personal and credit data, which are obligatory in each e-Contract.

The next stage, ($stg_3$), is assigned with the main body of an e-Contract, which is the negotiation of the e-Contract terms. These terms, mainly, refer to the terms of use and payment of the product or service under negotiation. Thus, after the negotiation and agreement of these terms, the procedure moves to the $stg_4$ stage. In this stage, the e-Contract is approved and all the extra necessary data are sent. These data may include, among others, technical details, access and credentials, depending on the scope of the e-Contract.

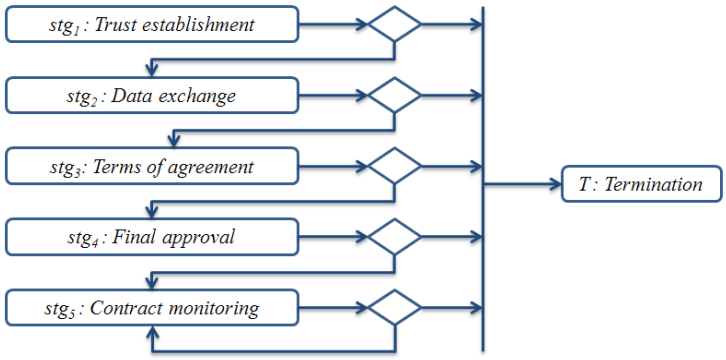


**Fig. 1.** The overview of the workflow methodology.

Finally, the above agreed procedure reaches the $stg_5$ stage, the contract monitoring stage, during which the e-Contract's content is actually executed and monitored for situations which are out the negotiation phase and involve mechanisms for detecting contract violations, sanction enactment, etc. The overview of the proposed workflow methodology is illustrated in Fig. 1.

## 3 Knowledge-based Workflow Model

In this section, we present the integration of the above workflow methodology into a multi-agent knowledge-based framework, called EMERALD [9] [10], providing among others flexibility, reusability and interoperability of behavior between agents. The main advantage of this approach is that it provides a safe, generic, and reusable framework for modeling and monitoring e-Contract agreements, which could be used for various types of on-line transactions among agents. Furthermore, our framework is based on Semantic Web and FIPA standards, to maximize interoperability and reusability.

### 3.1 The EMERALD Knowledge-based Framework

In order to model and monitor the parties involved in the e-Contract negotiation, a customizable, knowledge-based agent model, called *KC-AGENTS*, is deployed. Agents that comply with this model are equipped with a Jess rule engine [11] and a knowledge base (KB) that contains environment knowledge (in the form of facts), behavior patterns and strategies (in the form of Jess production rules). Actually, the Jess KB represents the agent's internal policy and implements the workflow transition conditions of each negotiation step $cnd_k^l$ to the next. Examples will be given in Section 4. The use of the KC-AGENTS model offers certain advantages, like interoperability of behavior between agents, as opposed to having behavior hard-wired into the agent's code.

A short description of the abstract specification of this model [9] is presented below for better comprehension. The generic rule format is: *result* ← *rule* (*preconditions*). The agent's internal knowledge is a set of facts $F \equiv F_u \cup F_e$, where $F_u \equiv \{f_{u1}, f_{u2}, \ldots, f_{uk}\}$ are user-defined facts and $F_e \equiv \{f_{e1}, f_{e2}, \ldots, f_{em}\}$ are environment-asserted facts. The agent's behaviour is represented as a set of potential actions–rules $P \equiv A \cup S$, where $A \equiv \{a \mid f_e \leftarrow a(f_{u1}, f_{u2}, \ldots, f_{un}) \wedge \{f_{u1}, f_{u2}, ..., f_{un}\} \subseteq F_u \wedge f_e \in F_e\}$ are the rules that derive new facts by inserting them into the KB and $S \equiv C \cup J$ are the rules that lead to the execution of a special action. Note that special actions can either refer to agent communication $C \equiv \{c \mid ACLMessage \leftarrow c(f_1, f_2, \ldots, f_p) \wedge \{f_1, f_2, ..., f_p\} \subseteq F\}$ or Java calls $J \equiv \{j \mid JavaMethod \leftarrow j(f_1, f_2, \ldots, f_q) \wedge \{f_1, f_2, ..., f_q\} \subseteq F\}$.

In order to provide a standard communication interface between the Jess KB and the agents, this framework provides a number of Java methods that can be invoked via Jess production rule actions. In addition, the framework provides one more facility, the *AYPS*, a customizable procedure for the *yellow pages service*, both for registered and required services. Its most important feature is that the proper providers are inserted into working memory as Jess facts with a designated format.

Moreover, as agents do not necessarily share a common rule or logic formalism, it is vital for them to find a way to exchange their position arguments seamlessly. Thus, the framework proposes the use of *Reasoners*, which are actually agents that offer reasoning services to the rest of the agent community. This approach does not rely on translation between rule formalisms, but on exchanging the results of the reasoning process of the rule base over the input data. The receiving agent uses an external rea-

soning service to grasp the semantics of the rulebase, i.e. the set of conclusions of the rule base.

One of these Reasoners (here called *Reasoner*) is the *defeasible logic Reasoner*, based on DR-DEVICE [12], which furthermore assumes an OO RDF data model that treats properties as encapsulated attributes of resource objects, providing more compact representation and property indexing. DR-DEVICE supports two types of syntax for defeasible logic rules: a native CLIPS-like syntax and an OO RuleML [13]-compatible one. The latter deals with extensions regarding rule types, superiority relations among rules and conflicting literals, as well as constraints on predicate arguments and functions. Using the Reasoner, agents communicate with each other, overcoming the fact that they may not comprehend the logic of the other party.

*Defeasible reasoning* [14] was selected because of its simple rule-based approach for efficient reasoning with incomplete and inconsistent information. Defeasible reasoning can represent facts, rules as well as priorities and conflicts among rules. Such reasoning with conflicts is useful in many applications, such as security policies [15], business rules [16], e-contracting [17], personalization, brokering [18], bargaining and agent negotiations [19], [20], [21]. When compared to mainstream non-monotonic reasoning, the main advantages of defeasible reasoning are enhanced representational capabilities and low computational complexity.

Finally, EMERALD provides an independent agent (called *Timer*) that simulates a time service, in order to synchronize agent transactions, which is used in the following use case paradigm (section 4). Real time could be used equally well.

### 3.2 Implementing the Workflow Methodology on EMERALD

Following EMERALD's specifications we commit to SW and FIPA standards, thus, we use the RuleML language [22] for representing and exchanging agent policies and e-contract clauses ${}_iC_k^l$, since it has become a de facto standard and it is very close to the RIF [23] emerging standard for SW rules. In addition, we propose the use of the RDF model for data representation both for the private data ${}_jD_k^l$ included in agent's $P_j$ internal knowledge and the results ${}_jE_k^l$ generated during the negotiation steps. The overview of the above proposal is illustrated in Fig. 2.

The agent $P_j$, in order to start an e-Contract negotiation process with $P_i$, asks the AYPS service for the latter's default call-for-negotiation requested value ($C_0$), required formalisms/languages, etc. Thus, $P_j$ sends a call-for-negotiation message (ACL message with *REQUEST* communication-act) to $P_i$ containing $C_0$. $P_i$ examines the new request and sends back a *REQUEST* message containing part of his clauses ${}_iC_k^l$ (in RuleML format), waiting for $P_j$'s reply or a termination. $P_j$, on his behalf, evaluates the receiving ${}_iC_k^l$ clauses using his own private data ${}_jD_k^l$ (in RDF format) and informs $P_i$ with a new message (ACL message with *INFORM* communication-act) containing the results ${}_jE_k^l$ (in RDF format).

Generally, the negotiation processes is a sequence of exchanged ACL messages; (both) parties use messages with *REQUEST* communication-act in order to ask for valid information and *INFORM* in order to reply. The process can either end success-

fully with an agreement between the parties or terminate at any step due to disagreement.

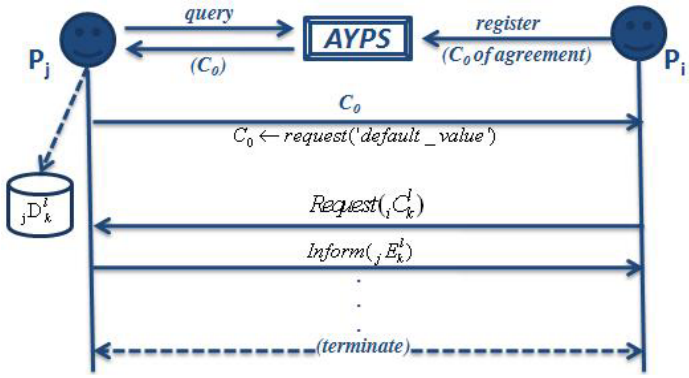


**Fig. 2.** The workflow implementation.

## 4 Use Case: The Wine Club

A Wine Club contract negotiation scenario is presented in order to illustrate the usability of the approach that involves two parties: a) the Wine Club, represented by its agent, that offers a variety of wines, and b) a customer who wishes to become a member of the above Wine Club. In addition, there are two extra agents involved: c) the *Reasoner* an independent third-party service that is responsible for conducting inference on defeasible logic rule bases and produces the results as an RDF file, d) the *Timer* an independent agent that simulates a time service, in order to synchronize agent transactions according to the contract's time schedule.

### 4.1 The Scenario Overview

First of all, the customer wants to be a member of a wine club, hence, uses the *AYPS* service (via sending an ACL message with *REQUEST* communicate-act) in order to find the appropriate Wine Club, which is registered to the directory. The *AYPS* sends back (via ACL message with *INFORM* communicate-act) the available providers with their default requested value ($C_0$). Hence, the customer finds the appropriate wine club service and sends a subscription request to it, namely a *REQUEST* message containing $C_0$ (call-for-negotiation). Since the Club has some terms in order to provide the service, provided by an e-Contract, at this point, a contract negotiation procedure between the Customer and the Wine Club, that follows the aforementioned methodology, begins (Fig. 3).

Following the context of the first stage ($stg_1$), both the Wine Club's agent, and the Customer have to provide sufficient evidence in order to certify that they can trust each other. Due to Customer's preferences this Wine Club service has to be a member of a trusted third-party organization, such as the *Best Business Bureau* [24], which

guarantees that the service satisfies the standard criteria. On the other hand, since it is forbidden to sell alcohol to people under the age of 18, the Customer has to provide credentials that certify his age. Thus, both reveal their credentials, successfully establishing trust among them and, thus, move to the next stage ($stg_2$).

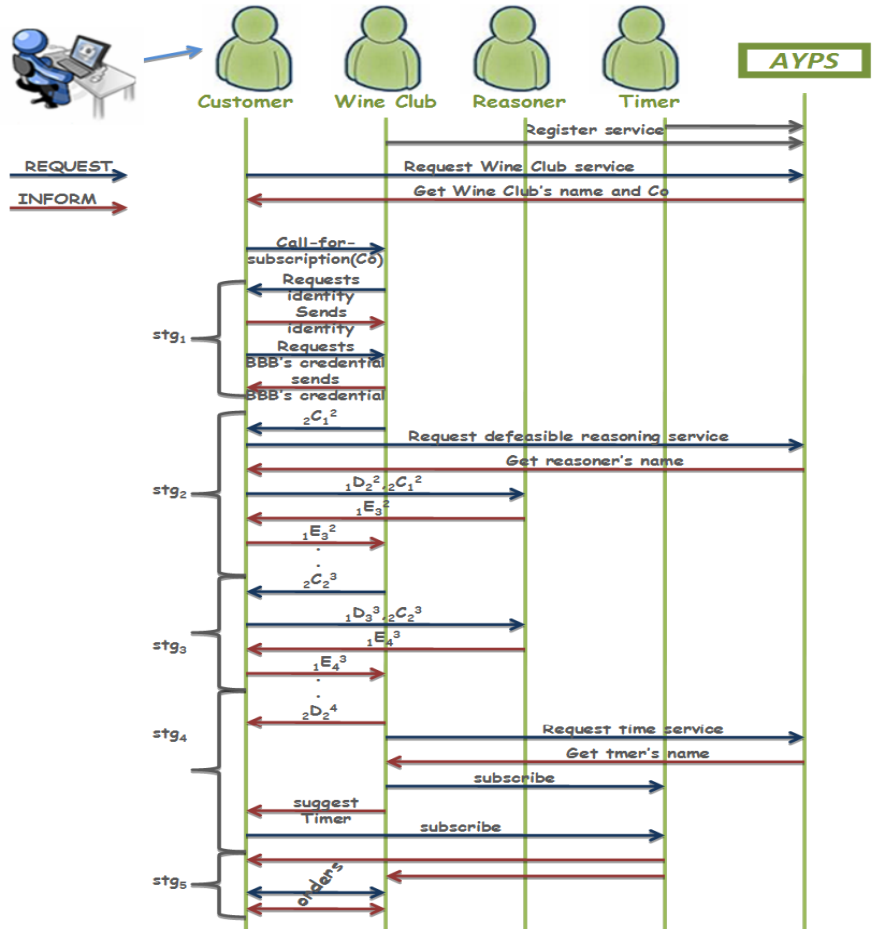


**Fig. 3.** The scenario overview.

At the first step of this stage, the Wine Club agent sends part of the e-Contract's clauses ($_2C_1^2$), in defeasible logic, requesting the Customer's personal data. This request includes among others, the Customer's name and e-mail, the credit card number and its date of expiration. The Customer, on the other hand, is willing to reveal a part of his personal data, but internally uses a different type of logic and cannot directly process Wine Club's defeasible logic requirements. Thus, an appropriate defeasible logic *Reasoner*, a trusted third-party reasoning service, is requested, which is retrieved from the directory service (*AYPS*). The Customer communicates with the

*Reasoner*, providing both his personal data (${}_1D_2^2$), in RDF, and the Club's arguments (${}_2C_1^2$), in defeasible logic and in a RuleML format, and waits for a reply. The *Reasoner* conducts inference on these arguments and data and produces the results as an RDF file. Thus, the Customer sends these results, namely his personal information that can be sent (${}_1E_3^2$), to the Wine Club agent.

Eventually, the current stage ends successfully and parties move to the next one ($stg_3$). As soon as the Club's agent receives the required data (${}_1E_3^2$), it responds with another part of the e-Contract (${}_2C_2^3$), a set of clauses represented in defeasible logic that contain the characteristics of the three available categories of subscription.

The three categories are the "Gold Customer", the "Silver Customer" and the "Bronze Customer", given in Table 1. The Customer communicates again with the *Reasoner*, providing him with both the RuleML file containing these clauses (${}_2C_2^3$) and the RDF file which contains his preferences (${}_1D_3^3$), such as the minimum price of an order (e.g. 500€) and order frequency (e.g. 1 per four months). Afterwards, the *Reasoner* sends the result (${}_1E_4^3$), which suggests the *Silver Customer* category. Thus, the Customer selects the most suitable category ending up the current stage. Eventually, acting in the context of the $stg_4$ stage, the Wine Club agent saves all the data concerning this customer and approves the subscription by sending him the catalogue of the available Wines (${}_2D_2^4$).

**Table 1.** The characteristics of each customer category.

| | |
|---|---|
| ***Bronze Customer*** | Order amount ≥ 200€& Order frequency ≥ 1 /4 months (at least 1 order per 4 months) & Suspension = 1(in a row) & Delivery_time = 7 days & Discount = 2% (in case of lack in order's products) & Return_fee = 10% |
| ***Silver Customer*** | Order amount ≥ 400€& Order frequency ≥ 1 /4 months & Suspension = 1(in a row) & Delivery_time = 7 days & Discount = 4% (in case of lack in order's products) & Return_fee = 8% |
| ***Gold Customer*** | Order amount ≥ 600€& Order frequency ≥ 1 /3 months & Suspension = 2(in a row) & Delivery_time = 5 days & Discount = 8% (in case of lack in order's products) & Return_fee = 6% |

At this stage a temporal dimension exists, since the customer has to order once per month (*Silver Customer*); both the Wine Club and the customer have to comply with the environment's common time representation scheme. Thus, the Club's agent finds a suitable time-agent (*Timer*), via the AYPS service, makes a subscription to this service and proposes the service to the customer. The customer, on his behalf, accepts the proposal and subscribes to the Timer's service. Finally, their interaction moves to the $stg_5$ for the agreed time period, carried out the regular orders, or until a break off.

### 4.2 Contract Terms and Information Specifications

Both the customer and the Wine Club's agent comply with the KC-Agents model (section 3), thus, they are equipped with a Jess rule engine and a Jess KB. Following this generic specification, the Wine Club agent's description contains facts and rules.

$F_u^{wc} \equiv \{categories\}, F_e^{wc} \equiv \{timer\_name\}$
$C^{wc} \equiv \{(ACLMessage\ (communicative\text{-}act\ REQUEST)$
$(sender\ Wine\_Club)(receiver\ Customer)(content\ categories))$
$\leftarrow requestCustomerCategory\ (\text{“}COND\text{”})\}$
$J^{wc} \equiv \{\text{“}triples\text{”} \leftarrow (bind\ ((new\ java\_class)\ getCustomerCategory\text{“}COND\text{”}\ ))\}$

Fact *categories* represent part of his internal knowledge and stand for the *categories characteristics*; part of the associated RuleML file is presented in Fig. 4.

```
<rulebase ...>
   <_rbaselab><ind type="defeasible">categories</ind></_rbaselab>
   <imp>
      <_rlab ruleID="r1" ruletype="defeasiblerule"/>
      <_head>   <atom>
                   <_opr><rel><ind>customer-type</ind></rel></_opr>
                   <_slot name="type"><ind>bronze</ind></_slot>
                </atom></_head>
      <_body>
          <atom>
             <_opr><rel><ind>order-pref</ind></rel></_opr>
             <_slot name="amount">
                <_and>
                   <var>a</var>
                   <function_call name="&ge;">
                      <var>a</var>
                      <ind>200</ind></function_call>
                </_and></_slot>
             <_slot name="frequency">
                <_and>
                   <var>f</var>
                   <function_call name="&ge;">
                      <var>f</var>
                      <ind>0.25</ind></function_call>
                </_and></_slot>
             ...
          </atom></_body></imp>
...
</rulebase>
```

**Fig. 4.** Part of the Wine Club's RuleML containing categories characteristics ($stg_3$).

Fig. 4 represents part of the Wine Club's first clause ($stg_3$) that implements the categories' characteristics (Table 1). Based on these clauses, presented below (in defeasible logic), and his personal preferences (Fig. 5), the customer is able to select the most suitable category. All three rules derive a positive conclusion, only one of which must be true, according to the constraint of the last line (conflicting literals).

$r_1$: *order-pref*(*Amount*,*Frequency*,*Suspension*,*Delivery_time*,*Discount*,*Return_fee*),
*Amount*≥*200, Frequency*≥*0.25, Suspension*≤*1, Delivery_time*≥*7,*
*Discount*≤*2*,*Return_fee*≥*10*

⇒ ***customer-type(type→bronze)***

$r_2$: *order-pref(Amount,Frequency,Suspension,Delivery_time,Discount,Return_fee),*
*Amount≥400, Frequency≥0.25, Suspension≤1, Delivery_time≥7,*
*Discount≤4, Return_fee≥8*

⇒ ***customer-type(type→silver)***

$r_3$: *order-pref(Amount,Frequency,Suspension,Delivery_time,Discount,Return_fee),*
*Amount≥600, Frequency≥0.33, Suspension≤2, Delivery_time≥5,*
*Discount≤8, Return_fee≥6*

⇒ ***customer-type(type→gold)***

$r_3 > r_2, r_2 > r_1, r_3 > r_1$

$C(customer\text{-}type(X)) = \{\neg\ customer\text{-}type(X)\} \cup \{\ customer\text{-}type(Y) \mid Y \neq X\ \}$

```
<rdf:RDF ...>
   <order-pref rdf:about="...">
      <amount rdf:datatype="&xsd;integer">300</amount>
      <frequency rdf:datatype="&xsd;float">0.3</frequency>
      <suspension rdf:datatype="&xsd;integer">1</suspension>
      <delivery_time rdf:datatype="&xsd;integer">10</delivery_time>
      <discount rdf:datatype="&xsd;integer">2</discount>
      <return_fee rdf:datatype="&xsd;integer">10</return_fee>
   </order-pref>
   . . .
</rdf:RDF>
```

**Fig. 5.** Part of customer's personal data (RDF).

Fact *timer_name* (the Timer's name) is added by the AYPS. Rules "*requestCustomerCategory*" and "*getCustomerCategory*" ($stg_3$) comprise part of the agent's behavior ($_2cnd_k^3$); part of the associated Jess file is presented in Fig. 6.

```
    ....
(defrule requestCustomerCategory "request customer's category"
   (personal_info received);stage 2 completed
   (MyAgent (name ?n))
   (customer ?x) (categories ?cc)
   =>
   (send (assert (ACLMessage (communicative-act INFORM) (sender ?n) (receiver ?x)
                                                              (content ?cc)))))
(defrule getCustomerCategory "get category"
   (customer ?x)
   (customer_personal_info ?x ?p)
   ?z<-(ACLMessage(communicative-act INFORM)(sender ?x)(content ?c))
   (test (or (eq ?c bronze)(eq ?c silver)(eq ?c gold)))
   =>
   (bind ?tt (new Basic))  (bind ?str (?tt extractTriples ?p))
   (batch ?str)))

(defrule find_name
   (triple (subject ?x) (predicate rdf:type) (object sendable))
   (triple (subject ?x) (predicate CustomersName) (object ?name))
   =>
   (assert(name ?name)))
...
```

**Fig. 6.** Part of the Wine Club's behavior in Jess ($_2cnd_k^3$).

Fig. 6 presents three of the Wine Club agent's behavior rules; the "*requestCustomerCategory"* rule, as soon as $stg_2$ is completed successfully, sends his clauses re-

lated to customer categories to the customer and waits; the *getCustomerCategory"* rule receives customer's reply, checks the selected category (bronze, silver, gold) and extracts the customer's personal data in RDF triples. Finally, the *"find_name"* rule finds the customer's name using the extracted triples in order to enter this customer in his register.

Similarly, the customer agent's description contains, among others, a fact *personal_data* which is part of his internal knowledge and represents his personal data. Moreover, due to the dynamic environment (AYPS is constantly updating the environment), new facts with the Wine Club's agent name (*Wine_Club*) are added to the working memory. Agent behavior is represented by rules that implement the workflow transition conditions of each negotiation step $cnd_k^l$ to the next; two of these are the "*request*" and "*read*"; the former is used for communication and the latter for Java method calls.

$F_u^{cust} \equiv \{personal\_data\}, F_e^{cust} \equiv \{Wine\_Club\}$
$C^{cust} \equiv \{(ACLMessage\ (communicative\text{-}act\ REQUEST)$
$(sender\ Customer)\ (receiver\ Wine\_Club) \leftarrow request\ (\text{"}Co\text{"})\}$
$J^{cust} \equiv \{\ personal\_data\_string \leftarrow (bind\ ((new\ java\_class)\ read\ personal\_data)\}$

## 5 Related Work

A tightly related approach is the *DR-CONTRACT* [25] architecture for representing and reasoning on e-Contracts in defeasible logic. The architecture captures the notions relevant to monitoring the execution and performance of e-Contracts in defeasible logics. More specifically, the framework deploys the Defeasible Deontic Logic of Violation (DDLV) [17], expressed via a RuleML extension that combines deontic notions with defeasibility and violations. DR-CONTRACT takes as input a DDLV theory, downloads/queries input RDF documents, including their schemata, and translates the RDF descriptions into fact objects. Finally, the conclusions are exported as an RDF/XML document through an RDF extractor.

*SweetDeal* [26] is another rule-based approach to representing business contracts that enables software agents to create, evaluate, negotiate and execute contracts with substantial automation and modularity. SweetDeal builds upon the Situated Courteous Logic Programs (SCLP) knowledge representation in RuleML that includes prioritized conflict handling and procedural attachments for actions and tests. Process knowledge descriptions are also incorporated, represented as ontologies expressed in DAML+OIL, thereby enabling more complex contracts with behavioral provisions, especially for handling exception conditions (e.g., late delivery or non-payment) that might arise during the execution of the contract.

The $ER^{EC}$ framework [27] is another example for designing, modeling, enacting and monitoring e -Contract processes. The framework centers on an underlying meta-model that bridges the XML contract document with the Web Services-based implementation model of an e-Contract. The $ER^{EC}$ meta-model applies certain constructs for modeling e-Contracts, like clauses, activities, parties, exceptions, contracts and subcontracts. The framework also offers potential for automatic generating and dep-

loying workflows for e-Contract enactment, as well as facilities for analyzing what-if scenarios with respect to e-Contract clause violation.

Another work that automates price negotiations in e-commerce transactions using a rule-based implementation based on JESS utilized in the JADE multi-agent is presented in [28]. However, the focus of our work is e-contract negotiation / argumentation, rather than price negotiation.

Similarly, our approach considers e-Contracts in the Semantic Web, but it is, to the best of our knowledge, the only one that provides a workflow methodology, which models the procedure that can be followed for negotiating and sealing an e-Contract. This methodology divides the overall process of an e-Contract agreement in stages, providing a safe, reusable procedure for e-Contract agreements, which could be used for different types of on-line transactions among agents. Moreover, our approach takes into account trust issues ($stg_1$ stage), an extremely important issue on on-line transactions. In addition, it is the only approach that is embodied in a multi-agent platform, letting agents easily follow the necessary steps for sealing an e-Contract.

## 6 Conclusions and Future Work

This paper presented a policy-based workflow methodology for modeling and monitoring e-contract agreements among agents interacting in the Semantic Web. The proposed methodology consists of a sequence of steps, grouped to five stages, which could be used for different types of on-line transactions among agents. In addition, the integration of this methodology into a knowledge-based multi-agent framework is proposed, that provides among others flexibility and reusability. This paper also provides a use case that illustrates the technologies proposed.

As for future directions, it would be interesting to verify our model's capability to adapt to an even wider variety of scenarios. Furthermore, it would be interesting to integrate the representation formalisms of both the public and private agent policies, so that they can interoperate by sharing base and inferred predicates.

**Acknowledgements**

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communication and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence Program.

## 7 References

1. Berners-Lee T., et al.: The Semantic Web. Scientific American, 284(5):34-43 (2001)
2. Hendler J.: Agents and the Semantic Web. IEEE Intelligent Systems, 16(2):30-37 (2001)
3. Krishna P.R., Karlapalem K., Dani, A.R.: From Contracts to E-Contracts: Modeling and Enactment. In: Inf. Technol. and Management 6, 4 (Oct. 2005), 363-387 (2005)

4. Merz M., et al.: Supporting electronic commerce transactions with contracting services. International Journal of Cooperative Information Systems 7(4):249-274 (1998)
5. Chiu D.K.W., et al.: Workflow View Driven Cross-Organizational Interoperability in a Web-Service Environment. In: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web, p.41-56, May 27-28 (2002)
6. Daskalopulu A., Dimitrakos T., Maibaum T.: E-Contract Fulfilment and Agents' Attitudes. In: ERCIM WG E-Commerce Workshop on The Role of Trust in e Business, Zurich (2001)
7. Winsborough W., Li N.: Safety in Automated Trust Negotiation. Proceedings of the 2004 IEEE Symposium on Security and Privacy, IEEE Press, pp. 147-160, Oakland, CA (2004)
8. Lee A., Seamons K., Winslett M., Yu T.: Automated Trust Negotiation in Open Systems. Secure Data Management in Decentralized Systems, Springer (2007)
9. Kravari K., Kontopoulos E., Bassiliades N.: Towards a Knowledge-based Framework for Agents Interacting in the Semantic Web, 2009 IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology (IAT'09), Vol. 2, pp. 482-485, Milan, Italy (2009)
10. Kravari K., Kontopoulos E., Bassiliades N.: A Trusted Defeasible Reasoning Service for Brokering Agents in the Semantic Web, 3rd Int. Symp. on Intelligent Distributed Computing (IDC 2009), 13-14 October, Springer Berlin / Heidelberg, Vol. 237, pp. 243-248, Cyprus (2009)
11. JESS, the Rule Engine for the Java Platform, http://www.jessrules.com/
12. Bassiliades N., Antoniou G., Vlahavas I.: A Defeasible Logic Reasoner for the Semantic Web. IJSWIS, 2(1):1-41(2006)
13. Boley H.: An Introduction to Object-Oriented RuleML. EPIA'03, p. 4, Portugal (2003)
14. Nute D.: Defeasible Reasoning. 20th Int. Conf. on Systems Science. IEEE Press, pp. 470-477 (1987)
15. Li N., Grosof B. N. and Feigenbaum J.: Delegation Logic: A Logic-based Approach to Distributed Authorization. ACM Trans. on Information Systems Security, 6(1) (2003)
16. Antoniou G. and Arief M.: Executable Declarative Business rules and their use in Electronic Commerce. Proc. ACM Symposium on Applied Computing (2002)
17. Governatori G.: Representing Business Contracts in RuleML. In: Int. J. of Cooperative Information Systems, 14(2-3):181–216 (2005)
18. Antoniou G., Skylogiannis T., Bikakis A., Doerr M., Bassiliades N.: DR-BROKERING: A Semantic Brokering System. Knowledge-Based Systems, 20(1), pp. 61-72 (2007)
19. Governatori G. , ter Hofstede A. and Oaks P.: Defeasible Logic for Automated Negotiation. In: Proceedings of CollECTeR 2000. (2000)
20. Governatori G., Dumas M., ter Hofstede A. and Oaks P.: A formal approach to protocols and strategies for (legal) negotiation. Proc. ICAIL 2001, pp. 168-177 (2001)
21. Skylogiannis T., Antoniou G., Bassiliades N., Governatori G., Bikakis A.: DR-NEGOTIATE – A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies. Data & Knowledge Engineering, 63(2), pp. 362-380 (2007)
22. RuleML, http://ruleml.org/
23. RIF, http://www.w3.org/2005/rules/wiki/RIF_Working_Group
24. Best Business Bureau Organization, http://www.bbb.org/
25. Governatori G, Hoang D.P.: A Semantic Web Based Architecture for e-Contracts in Defeasible Logic. In: Adi A., Stoutenburg S., Tabet S. (eds.) Proceedings RuleML-2005, Springer-Verlag, LNCS 3791, pp. 145 - 159 (2005)
26. Grosof B.N., Poon T.C.: SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies and Process Descriptions. In: 12th WWW, pp. 340–349, ACM Press (2003)
27. Krishna P.R., Karlapalem K., Chiu, D.K.: An $ER^{EC}$ Framework for e-contract Modeling, Enactment and Monitoring. In: Data Knowl. Eng. 51(1):31-58 (2004)
28. Badica C., Ganzha M., Paprzycki M.L: Implementing Rule-Based Automated Price Negotiation in an Agent System, *J. of Universal Computer Science*, 13(2), pp. 244-266 (2007)