Automated reasoning in a Collaborative Problem Solving Process - A first approach.

María Clara Casalini and Guillermo Simari
Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur
Bahía Blanca, 8000, Buenos Aires, Argentina
{mcca,grs}@cs.uns.edu.ar

Abstract

Collaborative systems are becoming a widely used tool particularly among professional communities and research groups. They provide a suitable context for distributed people working on the same subject to concentrate their work in a shared place to which everyone has access and can keep updated. The Semantic Web, intended to provide well defined semantics and cooperation between machines and people offers a range of technologies that underpin the development of intelligent collaborative systems. Defeasible reasoning has been shown to appropriately model common-sense reasoning. This work presents a first approach to incorporating defeasible reasoning with a software agent that collaborates with humans in a collaborative problem solving process. The process relies in a shared knowledge repository which is also modeled to incorporate defeasible knowledge.

Keywords: defeasible reasoning, argumentation, semantic web, collaborative systems, knowledge repositories, collaborative problem solving

1 INTRODUCTION

During the last years there have been important advances around the Semantic Web; research in the area is extensive, new tools are being developed, and many applications are being constructed that implement the existing standards and technologies. However there is still a considerable shortage of applications that provide capabilities for reasoning on the web, partly due to the shortage of accepted standards and frameworks to facilitate the implementation. As the authors in [13] exhibit there is a wide use of Semantic Web technologies and standards, particularly in research communities. Its usage can also be appreciated in government initiatives as well as business and commerce efforts. However there is still an important need of a web of data with shared semantics and support for rules and inferencing -an intelligent web. A similar scenario can be described in the area of collaborative systems. The spread of virtual communities of every kind: organized or spontaneous, local or global, formal or informal, is exposing the need for systems capable of fulfilling the needs that the members of these communities have. One of the features that users could take advantage of is the possibility of having software agents -some systems called them *bots*- to collaborate as pairs in the diverse tasks that human users usually perform.

This work proposes the incorporation of defeasible reasoning in a collaborative system that was designed following Semantic Web standards. It shows the re-definition of a collaborative problem solving process that relies on a repository of knowledge based on Semantic Web technologies. The repository of knowledge is defined to incorporate defeasible knowledge and the problem solving process integrates the tasks performed by human participants with the participation of a software agent that collaborates automating parts of the process.

The article is organized as follows. Section 2 introduces related work on the knowledge areas relevant for this work: Semantic Web and its languages; and Defeasible Knowledge Representation, some formalisms and proposals. Section 3 presents the proposed repository of defeasible knowledge and Section 4 describes the problem solving process based on the presented repository. The problem solving process is described in terms of the rules defined for the implementation of a software agent that collaborates with the humans in the process, and the facts and rules that are created for every piece of knowledge that the users add to the repository. Finally Section 5 presents the conclusions and some ideas for future work.

2 BACKGROUND

This section presents the relevant concepts and knowledge areas for this work. The Semantic Web definition and objectives are briefly explained and two of the languages defined for the implementation of the Semantic Web are introduced: RDF [9] and OWL [10] both closely related with the models we describe for a repository of knowledge and collaborative problem solving. Next a brief description of the main aspects of the DeLP framework is given. The DeLP language and the argumentation formalism main components are also explained. A summary of the proposal presented in [1] to introduce rules -defeasible and strict- in the Semantic Web closes the section.

2.1 Semantic Web Languages

The Semantic Web was described in [2] as an extension of the current web, in which the focus is set on giving information a better-defined meaning, enabling the cooperation between machines and people. To achieve this goal, new and redefined web technologies were needed. Among other things it was predicted that it was necessary to define new languages for the web, software tools to understand them and use them, together with well defined practices and recommendations to standardize its use and enable true integration of the diverse applications. With this objective in mind a set of layers was defined, to be implemented with a stack of languages and tools such as XML [11] -the eXtensible Markup Language- and XML Schema [14], RDF -the Resource Description Framework- and RDF Schema [3], OWL -a Web Ontology Language- and SPARQL [12] -a query language.

The Resource Description Framework (RDF) is a framework for representing information in the Web. The framework includes the RDF language -a general purpose language designed for representing information on the web, and RDF Schema -the vocabulary description language for RDF. Information expressed using RDF consists in a collection of triples formed by a *subject*, a *predicate* and an *object*. The underlying structure is that of a graph, where each triple is represented by two nodes connected by an arc: the nodes are the subject and the object, which in turn are related by the predicate (the arc). Each triple represents a sentence that expresses the value (object) for a certain property (predicate) describing a particular resource (subject) -see Figure 1. The graph representation of RDF is useful when showing information to human users of the language, however to exchange information among applications RDF has a recommended XML serialization form (RDF-XML).

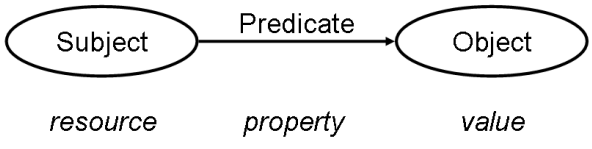

Figure 1: RDF graph notation.

To capture the semantics of data, an ontology layer was designed to be on top of the RDF layer. A special language to define and instantiate web ontologies was defined in the context of the Semantic Web Activity: the Web Ontology Language (OWL). An ontology includes descriptions of classes -to classify resources-, properties and their instances. The OWL formal semantics specifies how to derive logical consequences from the data present in the ontology, i.e. information not explicitly present in the ontology but entailed by its semantics [10]. Three sub-languages were defined for OWL, each with different expressiveness: OWL Lite, OWL DL and OWL Full. The Lite version supports classification hierarchies and simple constraints such as cardinality constraints with values 0 or 1. The second version was called OWL DL for its correspondence with Description Logics. OWL DL gives support for users who need more expressiveness than OWL Lite without losing computational completeness and decidability. For maximum expressiveness but with no computational guarantees, the Full version of OWL is provided. This version gives more freedom to the user, allowing for instance a class to be treated at the same time as a collection of individuals and as an individual itself.

2.2 Defeasible Knowledge Representation

2.2.1 Defeasible Logic Programming

Defeasible Logic Programming (DeLP) is a formalism that combines results of logic programming and defeasible argumentation. This formalism provides the possibility of representing information in the form of weak rules in a declarative manner and a defeasible argumentation inference mechanism for warranting the entailed conclusions [6]. Contradictory knowledge can be represented since it allows strong negation in the head of program rules. DeLP incorporates an argumentation formalism for the treatment of contradictory knowledge that allows the identification of the pieces of knowledge that are in contradiction. A dialectical process is used for deciding which information prevails. A warrant procedure implements such dialectical analysis.

DeLP Language. The DeLP language is defined in terms of three disjoint sets: a set of facts -literals: atoms or strict negation of atoms-, a set of strict rules and a set of defeasible rules. A strict rule is an ordered pair, denoted $Head \leftarrow Body$, where the head is a literal and the body is a finite non-empty set of literals. A defeasible rule is an ordered pair Head $-\prec$ Body., where the head is a literal and the body is a finite non-empty set of literals. Strict rules are different from defeasible rules because the latter ones represent defeasible knowledge i.e. information that may be used if no other information is known that contradicts it. The syntactic distinction between them is achieved by the symbol $\prec$.

A Defeasible Logic Program ($de.l.p.$) $\mathcal{P}$ is a possibly infinite set of facts, strict rules and defeasible rules. In a program $\mathcal{P}$, also denoted as (Π, Δ), Π is the subset of facts and strict rules and Δ is the subset of defeasible rules.

Let $\mathcal{P} = (\Pi, \Delta)$ be a de.l.p.and L a ground literal. A *defeasible derivation* of L from $\mathcal{P}$, denoted $\mathcal{P} \mid\sim L$, consists of a finite sequence $L_1, L_2, \ldots, L_n = L$ of ground literals, and each literal L_i is in the sequence because:

(a). L_i is a fact in Π, or

(b). there exists a rule R_i in $\mathcal{P}$ (strict or defeasible) with head L_i and body $B_1, B_2, \ldots, B_k$ and every literal of the body is an element L_j of the sequence appearing before L_i ($j < i$)

Defeasible derivations can use both defeasible and strict rules, or only one kind of rule. Strict derivations instead, only use strict rules and facts, as the following definition explains.

Let $\mathcal{P} = (\Pi, \Delta)$ be a de.l.p.and h a literal with a defeasible derivation $L_1, L_2, \ldots, L_n = h$, h has a strict derivation from $\mathcal{P}$, denoted $\mathcal{P} \vdash L$, if either h is a fact or all the rules used for obtaining the sequence $L_1, L_2, \ldots, L_n$ are strict rules.

Defeasible Argumentation. In the following lines we present some of the main concepts relevant to the argumentation formalism of the DeLP framework. Queries in DeLP are supported by arguments; let h be a literal, and $\mathcal{P} = (\Pi, \Delta)$ a de.l.p., $\langle \mathcal{A}, h \rangle$ is an argument structure for h, if $\mathcal{A}$ is a set of defeasible rules of Δ, such that:

1. there exists a defeasible derivation for h from $\Pi \cup \mathcal{A}$,
2. the set $\Pi \cup \mathcal{A}$ is non-contradictory, and
3. $\mathcal{A}$ is minimal: there is no proper subset $\mathcal{A}'$ of $\mathcal{A}$ such that $\mathcal{A}'$ satisfies conditions 1 and 2.

An argument $\mathcal{A}$ for h is a minimal non-contradictory set of defeasible rules, obtained from a defeasible derivation for a given literal h. This literal is also called the 'conclusion' supported by $\mathcal{A}$. A *sub-argument structure* of $\langle \mathcal{A}, h \rangle$ is an argument structure $\langle \mathcal{B}, q \rangle$ such that $\mathcal{B} \subseteq \mathcal{A}$.

An argument structure $\langle \mathcal{A}_1, h_1 \rangle$ *counter-argues*, *rebutts*, or *attacks* $\langle \mathcal{A}_2, h_2 \rangle$ at literal h, if and only if there exists a sub-argument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that h and h_1 disagree. Two literals h and h_1 disagree if and only if the set $\Pi \cup \{h, h_1\}$ is contradictory.

2.2.2 Defeasible Rules for the Semantic Web

The use of rules for providing reasoning capabilities to the Semantic Web is discussed in the work presented in [1]. The authors show how description logics and definite Horn logic are not reducible to each other, both having the ability to express things that the other cannot express. While it seems natural and highly useful to add rules on top of web ontologies (RDF and OWL) this approach presents computational and linguistic difficulties. The work proposes the use of rules for the ontology, logic and proof layers of the semantic web. Their modeling of RDF(S) and OWL statements with rules (based on the work of [8]) is shown in Table 2. The authors exemplify the usage of strict as well as defeasible knowledge.

RDF(S) Statements		OWL Statements	
$Statement(a, P, b)$	$P(a, b)$	$C\ sameClassAs\ D$	$C(X) \rightarrow D(X)$
$type(a, C)$	$C(a)$		$D(X) \rightarrow C(X)$
$C\ subClassOf\ D$	$C(X) \rightarrow D(X)$	$P\ samePropertyAs\ Q$	$P(X, Y) \rightarrow Q(X, Y)$
$P\ subPropertyOf\ Q$	$P(X, Y) \rightarrow Q(X, Y)$		$Q(X, Y) \rightarrow P(X, Y)$
$domain(P, C)$	$P(X, Y) \rightarrow C(X)$	$transitivePoperty(P)$	$P(X, Y), P(Y, Z) \rightarrow P(X, Z)$
$range(P, C)$	$P(X, Y) \rightarrow C(Y)$	$inverseProperty(P, Q)$	$P(X, Y) \rightarrow Q(X, Y)$
			$Q(X, Y) \rightarrow P(X, Y)$
		$functionalProperty(P)$	$P(X, Y), P(X, Z) \rightarrow Y = Z$

Figure 2: RDF(S) and OWL constructs using rules.

2.3 Knowledge Repositories for Collaborative Systems

In previous works [5, 4] a model for a collaborative problem solving process in virtual communities of practice was presented. This process relies on a repository of knowledge, the body of information and resources collected and developed by the members of the community. The repository is structured

following the RDF language model; formed by three different sets of components: resources -the community assets-, properties -relations between resources and data to describe the resources- and statements -sentences expressing information about resources through their properties-. The resources contained in the repository are categorized according to a hierarchy of classes or resource types. Properties are binary relations of two kinds: with resource types as domain and range, and with a resource type as domain and a simple (predefined) value type as range. The first kind of property allows to express relations between different resource types while the second helps on describing the resource types with expressions using basic value types such as strings or numeric values. The last set of components -the statements- are triples formed by three elements: a subject, a predicate and an object. The subject is a resource, the element being described by the statement; the predicate is a property, what describes the subject; and the object is the value of the property or resource to which the subject is being related by the property.

The collaborative problem solving process was designed to provide members of online communities with computer support to help them share their knowledge and experience throughout the process of solving a new problem. The model of problem solving process presented is defined in terms of six steps, starting with the introduction of a new problem and finishing with the consolidation of its solution. The process relies in the contents and form of the repository and interacts with it by using the existing knowledge as well as enriching it with new. The six steps are:

i. problem registration,
ii. problem exploration,
iii. problem matching,
iv. solution design,
v. solution refinement and
vi. solution integration.

Problem registration is the triggering step and takes place when a member adds a new problem to the repository, inviting the community to work in collaboration to build a solution for it. The second step is the problem exploration, during this phase the people working on the process analyze the problem with the objective of discovering information about the problem that will help members understand it. Every piece of information about the problem is added to the repository in the form of statements; new resources and properties may also be created. The following step is called problem matching. In this step the new problem is compared against every other problem existing in the repository to find similarities that may guide the process of building the solution. A number of predefined criteria are used for the comparison. The fourth is the solution design step. A new resource is added to the repository to represent the solution being built, this resource will be the subject of the statements describing the solution. New statements are added to the repository to express the fact that the new solution resource is being built to solve that particular problem. Whenever it is considered possible, the problem is decomposed into sub problems to be solved by a similar process. After the solution resource has been created, the solution refinement starts. This fifth step is where the construction of the solution takes place. In the same way the problem was explored, the knowledge discovered and created during the construction of the solution is incorporated as new elements of the repository. New resources described by statements and possibly new properties to describe them are obtained as a result of this step. Finally the solution integration step concludes the process. When the solution construction can be considered finalized new statements expressing this fact are added to the repository and the statements added during the solution design are deleted.

3 DEFEASIBLE KNOWLEDGE REPOSITORY

The objective is to incorporate the possibility of expressing defeasible knowledge in the shared repository as well as the participation of agents in the problem solving process to automate reasoning. In order to do this it is necessary to perform a translation from the RDF(S) language used to model and express the repository content to rules in a language that allows possibly inconsistent and incomplete knowledge such as the DeLP language. In [1] and [7] the authors explain how to express RDF(S) and OWL constructs in the form of rules; particularly in [7] a translation is shown from OWL-DL to DeLP programs.

This section introduces the representation in DeLP language for each of the elements of the repository of knowledge presented in section 2.3. Figure 3 depicts the representation for each element of the repository and exemplifies each rule with possible real values of elements in a repository.

Element		**Rule**	**Examples**
Classes	(Resource Types)	1. C(X)	$book(X)$ $person(X)$ $string(X)$
	(SubClassOf relation)	2. C(X) $-\!\prec$ D(X).	publication(X) $-\!\prec$ book(X).
Properties	(Property domain)	3. C(X) $-\!\prec$ P(X,Y).	person(X) $-\!\prec$ author(X,Y). person(X) $-\!\prec$ surname(X,Y). book(X) $-\!\prec$ book_title(X,Y).
	(Property range)	4. C(Y) $-\!\prec$ P(X,Y).	book(Y) $-\!\prec$ author(X,Y). string(Y) $-\!\prec$ surname(X,Y). string(Y) $-\!\prec$ book_title(X,Y).
	(SubPropertyOf relation)	5. P(X,Y) $-\!\prec$ Q(X,Y).	title(X,Y) $-\!\prec$ book_title(X,Y).
Resources		6. $C(a)$	$person(res01)$ $book(res02)$ $string(saramagoJose)$ $string(EnsaioSobreACegueira)$
Statements		7. $P(a, b)$	$author(res01, res02)$ $surname(res01, Saramago)$ $book\_title(res02, EnsaioSobreACegueira)$

Figure 3: DeLP representation of the Knowledge Repository

The first and second rules show how classes (resource types in the repository) are modeled and how the `SubClassOf` relation between two classes is represented. Properties are defined using two rules, one to specify the domain class of the property and the other one to specify the range. A third rule is given to specify the relation `SubPropertyOf` between properties. The resources comprising the repository are modeled with the sixth rule which specifies the identifier representing the resource and the class to which the resource belongs. The last rule models statements specifying their predicate, subject and object.

4 PROBLEM SOLVING PROCESS

Solving problems is a process that human beings carry on sometimes in an unstructured manner and sometimes following very precisely defined steps. In any case the process requires from the problem solver to have the basic knowledge to support the construction of the solution, the ability to find out the right tools to use, and some amount of creativity and reasoning capabilities to "join the pieces together". However, the human solver could obtain great benefits from the automation of parts of the

process. An intelligent software agent can ease the job of the human problem solver by taking care of repetitive tasks and by providing automated reasoning. The software agent can present to the human solver the results from its work for him to accept or decline the new generated knowledge and -when accepted- to incorporate it to the shared repository. This section describes how this automation can take place in each step of the process.

Classes	**Properties**	
	Domain	Range
$person(X)$	thing(X) $\prec$ is_about(X,Y).	topic(Y) $\prec$ is_about(X,Y).
$practice\_area(X)$	person(X) $\prec$ expert_in(X,Y).	topic(Y) $\prec$ expert_in(X,Y).
$problem(X)$	report(X) $\prec$ best_practice(X,Y).	practice_area(Y) $\prec$ best_practice(X,Y).
$report(X)$	topic(X) $\prec$ is_relevant(X,Y).	practice_area(Y) $\prec$ is_relevant(X,Y).
$solution(X)$	person(X) $\prec$ can_help(X,Y).	problem(Y) $\prec$ can_help(X,Y).
$thing(X)$	report(X) $\prec$ recommended_report(X,Y).	problem(Y) $\prec$ recommended_report(X,Y).
$topic(X)$	solution(X) $\prec$ is_solving(X,Y).	problem(Y) $\prec$ is_solving(X,Y).
...	solution(X) $\prec$ solves(X,Y).	problem(Y) $\prec$ solves(X,Y).
	problem(X) $\prec$ sub_problem(X,Y).	problem(Y) $\prec$ sub_problem(X,Y).
	problem(X) $\prec$ depends_on(X,Y).	problem(Y) $\prec$ depends_on(X,Y).
	...	

Table 1: Knowledge Repository Examples of Content: Classes and Properties

1. Problem Registration

The problem solving process is started by a member that presents a new problem for the community to consider and analyze in collaboration. This step is called the Problem Registration and it consists of the member adding a new resource of type Problem to the repository. This is done by adding a new fact to the knowledge base:

A member adds a new problem called `problemIdXX`. The new fact is:
$problem(problemIdXX)$

2. Problem Exploration

After the problem has been added to the repository and can be identified as a resource, the next step is to acquire knowledge about the problem in order to provide members and the software agent with a proper base to develop a solution. As a result of this Problem Exploration step, the members will discover properties of the problem. Some of this properties will be already defined in the repository so the members will add the statements to assert that information. Other properties will need to be added to the repository. In any of these cases the properties relate the problem with other resources, for those that do not already exist at this stage in the repository, the corresponding fact to define them will be added. It may also be necessary sometimes to add new classes or resource types to the repository definition in order to be able to represent every discovered resource. While the members continue to add information about the problem, the software agent performs an analysis of this information: an automated reasoning job that seeks the discovery of information and generation of new knowledge from the existing in the repository. This task is performed by following a set of rules that are given to the agent.

The people working on the construction of the solution for $problemIdXX$ add new elements to the repository as new information about the problem is discovered. The software agent follows the given rules to discover useful information. The example shows the rules given to the agent. Once a member adds a sentence to state the topic of the problem ($is\_about(problemIdXX, topicT)$) the first rule indicates how to discover experts that might help in the process. A second rule is given to find reports on best practices for any practice area relevant for the topic of the problem.

can_help(E, P) $\prec$ *is_about(P,T); expert_in(E, T).*

recommended_report(R, P) $\prec$ *is_about(P,T); is_relevant(T, A); best_practice(A,R).*

3. Problem Matching

In the third phase, -the Problem Matching step- the information added to the repository concerning the new problem is compared against the information existing about the rest of the problems already added and probably solved in the repository. For every problem in the repository the agent will consider a set of queries designed to compare that problem with the problem under consideration. The results will be of use for the problem solvers during the next phase of the process, when the solution is designed. Finding similarities with other problems in the repository helps in the understanding of the problem and shows possibilities for reusing work.

Three rules are defined for comparing the existence of statements about two problems using a certain property: $aANDb$, $aANDNOTb$, $aANDbValues$. The first and second rules ensure the similarity (or difference) of the problems in terms of the property that describes them and the third ensures the similarity in terms of the property as well as the value such property takes for the problems. These three rules are used to define a set of rules that compare the complete descriptions of the problems. The first in this set of rules is $describedSubset(P1, P2)$ which represents the fact that the problem $P1$ has been described using a subset of the properties used to described the problem $P2$. This is only accepted if there is a strict derivation proving that there exists a property common to both problems and there is no property used to described $P1$ and not used to described $P2$. A second rule $describedSubsetProper(P1, P2)$ represents in a similar way the fact that the problem $P1$ has been described using a proper subset of the properties used to describe problem $P2$. A third rule $describedEq(P1, P2)$ represents pairs of problems with descriptions using the exact same set of properties. Three more rules similar to these are defined to compare the description of the problems considering in the comparison the values the properties take.

$aANDb(property, A, B) \leftarrow property(A, \_); property(B, \_)$
$aANDNOTb(property, A, B) \leftarrow property(A, \_); \sim property(B, \_)$
$aANDbValues(property, A, B) \leftarrow property(A, X); property(B, X)$

$describedSubset(P1, P2) \leftarrow aANDb(\_, P1, P2); \sim aANDNOTb(\_, P1, P2)$
$describedSubsetProper(P1, P2) \leftarrow aANDb(\_, P1, P2); aANDNOTb(\_, P2, P1); \sim aANDNOTb(\_, P1, P2)$
$describedEq(P1, P2) \leftarrow describedSubset(P1, P2); describedSubset(P2, P1)$

$describedSubsetV(P1, P2) \leftarrow aANDbValues(\_, P1, P2); \sim aANDNOTb(\_, P1, P2)$
$describedSubsetProperV(P1, P2) \leftarrow aANDbValues(\_, P1, P2); aANDNOTb(\_, P2, P1); \sim aANDNOTb(\_, P1, P2)$
$describedEqValues(P1, P2) \leftarrow describedSubsetV(P1, P2); describedSubsetV(P2, P1)$

4. Solution Design

The solution design step can take place when the problem has been analyzed and it can be understood enough to be able to decompose the problem into sub-problems, whenever this is possible. The objective is to divide the problem into sub-problems following the divide-and-conquer strategy. Each sub-problem is added to the repository in the same way as the original problem, and related to it by

creating a statement using the predicate $sub\_problem$. The relation among the different sub-problems is analyzed to detect possible dependencies. A new resource to represent the solution is created and the fact is added to the repository, together with a statement with predicate $is\_solving$ to relate the solution to the problem. If the problem was divided into sub-problems, the corresponding statements to relate them to the original problem are added as shown here.

Examples of statements to relate the problem with created sub-problems and a statement to express dependence between two problems are shown.
$is\_solution(solutionIdXX)$
$is\_solving(solutionIdXX, problemIdXX)$
$sub\_problem(subprobIdXX1, problemIdXX)$
$sub\_problem(subprobIdXX2, problemIdXX)$
...
$sub\_problem(subprobIdXXn, problemIdXX)$
$depends\_on(subprobIdXX3, subprobIdXX5)$

5. Solution Refinement

During the solution refinement step the members develop the solution to the problem. Every result obtained during this step is incorporated to the repository as a new element: resources, statements, properties, classes.

Examples of possible resources and statements discovered, created, and described during the solution refinement step are shown below.
$text\_document(documentXX)$
$software(applicationXX)$
$describes(documentXX, solutionIdXX)$

6. Solution Integration

The solution refinement step finishes once the solution construction has been completed. This event leads to the solution integration step, when the statement $is\_solving(solutionIdXX, problemIdXX)$ is removed from the repository and a new statement is added using the predicate $solves$.

A new statement relates the new solution to the problem, using the property $solves$ as predicate.
$solves(solutionIdXX, problemIdXX)$

After the sixth step is finished the solution to the problem has been developed and the information has been stored in the repository. These steps are followed for every problem that the members add to the shared repository. The process enriches the knowledge of the community significantly: the number of facts and rules is increased with every step of the problem solving process. This is not merely new information, it is new and valuable knowledge since many of these new rules and facts are new relations between existing resources of the repository.

5 CONCLUSIONS

The work presented an initial approach to incorporating automated reasoning in a collaborative problem solving process based on Semantic Web languages. Relevant material was gathered, studied and presented: the Semantic Web technologies and standards, appropriate logic and argumentation frameworks and the ongoing work on these subjects. It showed how to represent the knowledge repository

using rules and including strict as well as defeasible knowledge. The problem solving process was revised, explaining step by step how each phase in the process should be performed working with the redefined knowledge repository. The tasks performed by the human solvers did not change. The result is a simple but powerful model for the automation of the problem solving process that allows to incorporate the help of software agents as collaborators, easing the job of the humans and providing support for the knowledge discovery. While the work is in its initial stages, it provides a solid base for the next steps. Among the goals for future work are the improvement and extension of the crucial steps of the process from the point of view of automation such as the Problem Matching step, and the exploration of different methods for problem comparison.

REFERENCES

[1] Grigoris Antoniou and Gerd Wagner. Rules and Defeasible Reasoning on the Semantic Web. In *Rules and Rule Markup Languages for the Semantic Web*, volume 2876/2003 of *Lecture Notes in Computer Science*, pages 111–120. Springer Berlin / Heidelberg, 2003.

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[3] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[4] María Clara Casalini, Elsa Estevez, and Tomasz Janowski. Collaborative Problem Solving in Virtual Communities of Practice. - A Case Study in Disaster Prevention and Handling. In *XII Congreso Argentino de Ciencias de la Computación*, pages 206–217. RedUNCI, 2006.

[5] Maria Clara Casalini, Tomasz Janowski, and Elsa Estevez. A Process Model For Collaborative Problem Solving In Virtual Communities Of Practice. In *Seventh IFIP Working Conference on Virtual Enterprises.*, pages 343–350. Springer, September 2006.

[6] Alejandro Garcia and Guillermo Simari. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming*, 2002.

[7] Sergio A. Gómez, Carlos I. Chesñevar, and Guillermo R. Simari. Inconsistent Ontology Handling by Translating Description Logics into Defeasible Logic Programming. *INTELIGENCIA ARTIFICIAL*, 11(35):11–22, 2007.

[8] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic, 2003.

[9] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[10] Deborah L. McGuinness, Michael K. Smith, and Chris Welty. OWL web ontology language guide. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

[11] Jean Paoli, C. M. Sperberg-McQueen, and Tim Bray. XML 1.0 recommendation. first edition of a recommendation, W3C, February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[12] Andy Seaborne and Eric Prud'hommeaux. SPARQL query language for RDF. W3C recommendation, W3C, January 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[13] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

[14] Priscilla Walmsley and David C. Fallside. XML schema part 0: Primer second edition. W3C recommendation, W3C, October 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.