An Ontology Based Requirement Rule Consistency Checking Framework

Dr. Farheen Siddiqui
Dept. of computer science, Jamia Hamdard, Handard Nagar, New Delhi

Abstract: In this paper, we investigate how ontologies developed for use in Semantic Web technology could be used in automated consistency checking of application requirements. Ontology Driven Architecture allows developers to discover shareable domain models and knowledge bases from a variety of interrelated repositories and then wire them together with the remaining object-oriented components for user interface and control. Domain knowledge base(domain Ontology) captures domain concepts, relationships and rules. Requirements rules should note violate these rules or contradict the usual business behaviour. This paper suggests a rule editing and validation framework RECC (REquirement-rule Consistency Checking) that guides an analyst to enter (application specific) requirement rules. It rests on Semantic Web technologies together with reasoning engines, which operate with semantic representations. A practical validation of the approach by instantiating this framework with OWL and reasoning engines is presented here. When requirement rules are authored with RECC the acquired requirements would comply with both business needs and domain knowledge.

Key Words: Knowledge authoring, Semantic Web, Ontology rule editor, reasoning engines.

I. INTRODUCTION
Knowledge authoring has become a fundamental process in the current knowledge society, since it allows organizations and entities to obtain and manage valuable information when taking decisions. This process usually consists of three stages [32], involving domain experts and knowledge system administrators. This paper is particularly focused on the phase wherein an expert adds application specific knowledge (requirement rule and validates it .. The processes studied in this paper are based on the use of Semantic Web technologies [3,11]. The adoption of the Semantic Web overcomes the search and integration limitations of knowledge management systems [15]. More specifically, ontologies are adopted based on Description Logic (DL) [1, 2] as the representation (Fig 1) of the domain model in such processes. Knowledge models based on DL ontologies are usually divided into TBox (terminological) and ABox (assertion) components

![Diagram](image_url)

Fig 1: Knowledge models represented by means of DL ontologies with ontology rules.

The TBox contains the vocabulary and schema that define domain concepts, their properties, and the relationships (called roles in DL) among them. Hence, a concept represents a set of elements with similar characteristics (e.g., Students, StudyProgram, Instructor etc.), whereas roles symbolize binary relationships among elements (e.g., Student EnrollIn StudyProgram). Apart from these two atomic components, the DL language offers some axioms and restrictions over them to represent more complex domains (e.g., a RegularStudent is a specialization of the Student concept; a Student must have at least Enrollment). On the other hand, the ABox is populated with instances of these concepts and roles, representing a specific situation in the domain according to that schema. Let us see a simple example of an ontology rule. Suppose the domain model of a system which manages the task of a university system is represented by an ontology where Student and StudyProgram are concepts, and EnrollIn is a role that relates Student to their enrollment in a specific course. Moreover, Student has a specialization MCAStudent. The ABox of the ontology contains the assertions Student(Mark), StudyProgram(MCA) and EnrollIn(MarkMCA). Now, suppose that a system administrator wants to express that all students Enrolled in MCA are MCA students. This conditional statement can be defined through the knowledge rule by using the previous ontology elements as follows: Student(?x) StudyProgram(MCA) ^ EnrollIn(?x, MCA) - >MCAStudent(?x)

The main contribution of this paper is the development of a generic rule authoring system in a Semantic Web framework which includes the characteristics listed in the above paragraph. To this end, we have developed a framework RECC which consist of a GUI that guided the user through simple steps when editing rules. RECC enable a comfortable rule editing, testing, debugging and validation. RECC consists in a graphical front end which guides the user in the management application specific business rule. RECC is a stand-alone application aimed to graphically edit, test, debug and validate ontology rules in any domain. Therefore, the RECC is intended to be used in developing application that needs to work with Ontology, ontology rules and reasoning processes.

In this work we have focused on university scenario where there is a necessity of modeling and monitoring knowledge rules to illustrate the RECC functionalities. Particularly, this scenario is based on the management of information directed to manage task of university. The rest of the paper is structured as follows. The next section introduces the underlying elements on which RECC based, and then gives an abstract architecture of the rule authoring system.
Section III provides a description of the RECC architecture implementation, focusing on technical aspects of the authoring process. A scenario in which RECC is used to generate knowledge rules to control an intelligent building is exposed in section IV. Section V discusses several related tools for managing ontologies and rules. Finally, section VI summarizes our contribution and points out the future work.

II. RULE AUTHORIZING SYSTEM IN A SEMANTIC WEB FRAMEWORK

A. Motivation: A University scenario

In order to illustrate how RECC manages the authoring cycle of knowledge rules, it has been integrated into university system as a case study. The scenario corresponds to the management of a university. This university offers several study Programs in regular and online modes. The set of rules applicable to students are different depending on mode and Study Program. Also the rules governing student may change in every academic year. Thus university systems need to manage the edition and monitoring of rules, presenting an excellent field to test the authoring framework developed here. In particular, the scenario focuses on the management of Students enrolled in different study Programs. There are two types of rules: domain rules, which are applicable to all application of a specific domain and requirement rule which are applicable for the general operations a specific application under consideration. Suppose that Bob is a Student who has enrolled in a study Program with regular mode. The domain rule states that every student enrolled in Study Program must appear in examination. This domain rule will be represented as a knowledge rule, denoted by ExamRule henceforth (see section 4.1). The use of rules to detect inconsistencies in property values over elements in the domain is a desirable feature of these systems. If the relation between Student and Exam is missing, then RECC will be able to detect the missing relationship during requirement rule authoring. Thus, the generation of new knowledge through rules should be permitted in a simple manner.

Now consider a specific application requirement that a student with more a backlog paper is not allowed to appear in examination. This requirement rule will be represented as a knowledge rule, denoted by BacklogRule henceforth: Student(?x) ^ StudyProg(?p) ^ course(?c) ^ hasCourse(?p,?c) ^ EnrollIn(?x,?p) ^ exam(?e) ^ examHeld(?e,?c) ^ course(?d) ^ hasBacklog(?x, ?d) ^ notEqual(?c, ?d) ->Detained(?s)

Notice that these relationships are not modeled here as simple properties, but as concepts. The fundamental for this decision resides in that the types of such relationships could be used to classify them in different categories. Thus, EnrollIn is a special type of association that an student has with StudyProg. Eventually, observe that the domain and the range of these associations are described through roles.

Fig 2: The University scenario expressed in an OWL-DL ontology.

B. Generic rule authoring system architecture

Once the knowledge and rule models have been introduced, the next step developing the generic rule authoring system resides in establishing the architecture to deal with both of them. Figure 3 depicts the abstract architecture of the rule authoring system RECC. The core system (1) is formed by the RECC which encloses the methods for managing ontology rules; through GUI, a graphical interface of these tasks; the OWL API[4], which manages the ontologies representing the knowledge model it provides a set of methods for loading and managing OWL ontologies, together with a group of reasoning engines with different capabilities.

Fig 3: An abstract architecture view of RECC connected with its external resources.

OWL API is employed in the core of the system to obtain a working model from the domain ontology, not as a reasoning engine. RECC receives OWL ontologies and SWRL rules as input (2). RECC is in charge of SWRL rules, which can also be graphically edited, tested, debugged and validated in the GUI. Inference processes in RECC are accomplished by the combination of different reasoning engines (3). These engines can be distinguished according to the two types of inference that have been
explained in the introduction (see figure 1, the reasoning engines box), namely ontology and rule-based reasoning. Regarding ontology reasoning engines, there exist several proposals such as Pellet [31], Jena[6], Euler [30], Fact++ [25]. Likewise, there are several implementations available in the field of rule-based engines, such as SweetRules, JEOPS, JLisa, Prova, OpenRules, Jess, RDFExpert[9], Pellet and Jena.

III. ARCHITECTURE IMPLEMENTATION
The architecture previously presented in figure 3 has been implemented as framework for authoring of ontology rules. The following subsections present a description of all RECC framework components.

A. Details on implementation
Testing and debugging tasks in the rule authoring process demand that the developers get the full control in the execution of the rules, step by step, during the whole authoring session. Moreover, developers need to track the reasons for which rules have been fired in order to discover errors in the design of the rules. At the same time, both ontology and rule-based reasoning processes have to be taken into account in RECC to test and validate the edited rules.

B. RECC: Rule Authoring framework for software applications
RECC software Analyst which need knowledge rule authoring services. These services are directly implemented in the RECC as a set of methods to enter business rules, perform ontology inference processes, and test business rules, retrieve the inferred knowledge generated by ontology and rule reasoning processes, and accept/Reject rules during the authoring process.

The integration of the reasoning processes in RECC has been depicted in figure 6. It starts dividing the initial knowledge model into rules on one hand (SWRL), and ontology model (OWL) on the other hand. Next, the ontology model is loaded into the Pellet ontology reasoner producing a semantic enrichment of this ontology model as output. The reason for doing this separation between SWRL and OWL is for isolating the execution of the rules in order to get the full control over their execution for debugging and validating purposes. Then, ontology rules have to be transformed from the SWRL syntax managed in the RECC architecture to the specific rule format imposed by the rule reasoner in case it is necessary. Next, these rules and the semantically enriched ontology model are inserted in the rule reasoner. Finally, this reasoner infers new facts that could be grouped as rule-based knowledge as shown in figure 6. For example, the property Enrollment can only take one value for the same Student; therefore a semantic inconsistency occurs when this property takes more than one different value, for a specific application business needs. Thus, RECC is able to discover inconsistencies, and to notify it to the Analyst. Then, Analyst can decide to include the requirement rule again in the initial knowledge base in order to start a new step in the authoring process or just simply discard the requirement rule because a rule is violating the general domain restriction.

C. RECC GUI
RECC provides a GUI intended for being used by system administrators. All the functionalities involved in the knowledge rule authoring are offered here in a graphical mode. RECC provides a full control in the navigation
across the knowledge models managed in RECC the rule authoring process is performed hiding the details of the underlying SWRL syntax to the user. Analyst adds a requirement rule through a wizard that guides the creation or modification tasks. These tasks are executed in an easy and intuitive selection manner, where ontology elements (concepts, roles, individuals, etc.) are selected to the correspondent part of the rule structure displayed in the GUI. This structure is based on SWRL syntax, as previously exposed in section 2.2. To this end, the rule editor in RECC offers a complete vision on the domain ontology, where the user can navigate across all the information represented in it. Both antecedents and consequents of a rule are defined in the same manner: For example, according to the University scenario introduced in section 2, We can add a SWRL rule saying that an individual X from the Person class, which has Enrollment in study Program Y, belongs to a new class Student. Such rule is best described in the SWRL:

```
Person(?x) ^ StudyProg(?y) ^ EnrollIn(?x, ?y) -> Student(?x)
```

To edit this rule in RECC-GUI, the Analyst should first select the class Student and Study Program from the domain model listed in the class list box. The properties that are displayed in the property list box are strictly according to the objects properties in domain ontology having selected concepts either as range or domain. Then analyst can then select the EnrollIn Property (object property of Student) to the rule. The edition task is validated by the wizard, avoiding the appearance of syntactical errors during the rule definition.

Regarding consistency checking of requirement rule provided by RECC-GUI, it offers some options such as two objects can be related in head of a rule only if such relation exists in domain ontology. Also a requirement rule can be added only if it is not violating domain rule and already added requirement rule and domain rules. RECC informs about relationships in the antecedents of rule that are missing in the domain ontology or inconsistent with domain rule. After a rule has been entered, the analyst decides whether he wants to insert the inferred facts into the initial knowledge base in order to perform the next requirement rule insertion step or discard these facts totally or partially. As a result, RECC can be seen as a framework for an ontology rule editing, testing, debugging and validation.

The inference process depicted in figure 6 and performed by the selected reasoning combination is done just by a click action. Not only will this action provide new inferred knowledge and a consistence validation, but it also offers the different knowledge bases showing all the information involved in each step of the knowledge rule authoring process. This last feature confers an extra debugging power to RECC over Protégé [26] or Ontotrack [19], which do not implement it.

IV. USING THE RULE AUTHORING IN UNIVERSITY SCENARIO

The domain ontology only captures domain concepts and neglects domain-restricted rules, or shortly domain rules. Domain rules are the description of some definitions and restrictions of a business, which can be used to maintain business structure and control or influence business behavior. Those rules are the restrictions or constraints imposed on business behavior. They usually exist deep in the mind of users as important background information which is not easily documented. requirement rules are the description of some business process, which is to be used for a specific business structure. Hence requirement rule vary for different application of same domain (same domain rules). If analysts are not familiar with the domain, especially the domain rules, they would model a requirement rule that violates domain rules and hence making contradiction to the usual business behavior. Therefore, it is necessary to model requirement rules for application to achieve agreement not only on the domain concepts but also without violating the domain-restricted rules. We can capture all rules together in the application rules base. The later can be used to check if any requirement rule is inconsistent with domain rule. This scenario is used here to illustrate the entire knowledge authoring process described in the introduction of the paper, and it specially focused on the management of rules. Regarding the first stage, dedicated to the acquisition of knowledge, the Analyst has modeled the education domain by means of an OWL ontology based on the DMTF-CIM standard, called OWL-CIM [8]. This OWL ontology represents the concepts as partially shown in the TBox of figure 2. A complete vision of the scenario is depicted in figure 7. The creation of the OWL-CIM ontology that represents this specific domain has been performed in Protégé [26], a broadly used OWL editor. The specific scenario involving Student and the Exam management (partially represented in the ABox of figure 2) has been modeled following the same idea. The OWL files of this scenario exposed in figure 7 has been distributed within the RECC framework, as a case of study. It describes a university offering Study Programs in different Departments. For simplicity, just the two Department involved in the scenario, Computers and Management, has been depicted. The composition relationship has been modeled using the hasStudyProg association (see section 2.2). Each StudyProg contains in turn some Courses (through the hasCourse association).
S is a student of the university whose enrolled StudyProg is MCA (by means of the EnrollIn association). MCA is offered in Computers Department. Each Study Program conducts examination for the courses it contains. The Student is linked to Exam by the AppearIn association and course is linked to Exam by the HeldFor association. From the domain ontology we can know the following knowledge: Every student who is enrolled in a shall appear in exam for that study program corresponding rule:

**Exam Rule:**

\[
\text{Student}(?x) \land \text{StudyProg}(?p) \land \text{course}(?c) \\
\land \text{hasCourse}(?p, ?c) \land \text{EnrolledIn}(?x, ?p) \\
\land \text{exam}(?e) \land \text{examHeld}(?e, ?c) \rightarrow \text{Examinee}(?s)
\]

Particularly, university scenario deals with the management of Students and Instructor. On one hand, a domain rule states that every enrolled student appear in exam This rule has been recorded as domain rule in a domain rule base. Contrarily, the policy of a particular university states that student must not appear in an exam if he is having a pending backlog course. This rule is added to the by a system administrator through authoring process of RECC.

**Backlog Rule:**

\[
\text{Student}(?x) \land \text{StudyProg}(?p) \land \text{course}(?c) \\
\land \text{hasCourse}(?p, ?c) \land \text{EnrolledIn}(?x, ?p) \\
\land \text{exam}(?e) \land \text{examHeld}(?e, ?c) \land \text{course}(?d) \\
\land \text{hasBacklog}(?x, ?d) \land \text{notEqual}(?c, ?d) \rightarrow \text{Detained}(?s)
\]

where examinee and detained are defined to be disjoint classes in domain ontology.

This scenario is used here to illustrate the entire knowledge authoring process described in the introduction of the paper, and it specially focused on the management of rules. Regarding the first stage, dedicated to the acquisition of knowledge, the system administrator has modeled the University domain by means of OWL ontology. This OWL ontology represents the concepts as partially shown in the figure 2. The creation of the OWL ontology that represents this specific domain has been performed thanks to Protégé—a broadly used OWL editor.

### A. Rule authoring process

In order to show an example of the rule authoring process, suppose that a analyst is responsible of editing the different domain restrictions by means of rules. To this end, the analyst uses a kind of templates which gathers such users’ preferences to convert this information into rules. As a result, the analyst defines a rule that a student appear in exam of the course that belong to study Program in which student is enrolled with. This rule is called ExamRule and it is given below in SWRL abstract syntax. examrule can be read as follows: if a student is enrolled in a study Program with course ,and exam is held for that course and the student, then student appear in exam.

### B. Rule testing, debugging and validation tasks

After defining ExamRule, it is saved in rule file containing domain rules by clicking on the button “Save Rules.” Analogously, the analyst uses RECC to insert the requirement of the particular university as a rule, i.e. BacklogRule. Furthermore, in this scenario the RECC of the system administrator is also configured to automatically execute an inference process whenever a new rule is added through RECC, as for example if a student’s enrolled d in study program MCA with course Web Technology will be
Examinee for exam held for web technology. Now if s has a pending backlog other than web technology then s will be detained. Since examinee and detained are disjoint concepts in the underlying domain model, an inconsistency in the requirement is detected by RECC. The inference process provide all the inferred facts produced by the both ontology and rule reasoning. The Analyst could perform some actions. On one hand, he can decide to insert the inferred information into the system passing to the next step in the debugging session. On the other hand, he can decide to discard totally or partially the inferred information and to repeat the inference process changing any rule definition or enabling/disabling totally or partially some rules in order to test and validate the correctness of the rules available in the system. In this case, the RECC framework has detected an inconsistency in the knowledge base due to the existence of a conflict between the inferred facts according to ExamRule and BackLogRule. Both rules force the S to be Examinee and detained at same time. This situation violates the disjoint ness of Examinee and detained classes. RECC is able to detect this conflict and offers mechanisms to solve it manually. These mechanisms consist of the selective deletion of one of the conflicting facts.

V. RELATED WORK
As the popularity of the Semantic Web has rapidly increased, several ontology tools has been developed at the same time. Prot´eg´e [26] is a famous ontology tool with an OWL plug-in that allows the user to define her own ontologies, and to export them into a variety of formats including RDF(S), OWL, and XML. It also supports the edition and execution of SWRL rules [27]. SWOOP [28] is an IDE for developing ontologies, based on a Web browser interface. Hence, it allows browsing through hyper-links, which can be considered as an initial idea of showing ontologies in an intuitive way to the user. This tool has demonstrated to be useful for ontology debuging. However, it does not permit to debug or validate rules. AEGONT [24] is an ontology development environment on .NET framework, whose major innovation lies in the Rule and Query Views, although they both are not fully functional yet. SWeDE [29] (Semantic Web Development Environment), an extensible framework built on the Eclipse IDE including an OWL editor with features like syntax highlighting, autocompletion, and error-detection. It also integrates existing tools like the OWL Validator and DumpOnt (an ontology visualizer). The second one is RuleVISor [20], an alpha-tested rule editor. Also Pronto [17] is a reasoning engine with probabilistic reasoning support that may be included in RECC framework. Nevertheless, they lack a debugging mechanism and finding inconsistency between domains restricted rule and requirement rule.

VI. CONCLUSIONS
This paper presents a framework to manage the authoring process of rules in knowledge systems based on Semantic Web technologies. The knowledge model of these systems is normally given by means of ontologies. From these ontologies it is possible to define production rules (i.e., “if-then”) in order to describe requirements, in a natural and straightforward manner.

In this paper, RECC is presented in the form of a GUI. The management of ontology elements, the retrieval of the inferred facts generated by ontology and rule-based reasoning, or the activation/deactivation of rules during the inference process is some of the RECC features. On the other hand, RECC is a standalone application for users such as Analyst which offers the same operations to model requirement in form of rules that are consistent with underlying domain ontology. Once new rules have been created, they can be tested by means of the facilities incorporated in RECC. The platform which performs the inference process over the knowledge model is based on Jena and Pellet reasoning engines, although it may effortless be extended with new reasoner capabilities (e.g. fuzzy inference). As for debugging and validation, both syntactic and semantic checking of rule definitions has also been included in the RECC framework. The former avoids ill-formed rules, by warning the user if the rule is being bad defined. The latter detects knowledge conflicts among rules, which usually are complicated to discover. These conflicts arise because of numerous causes: contradictory consequents, ontology axiom violation, etc. The conflict is then reported to the user, and besides a manual solving mechanism is provided.

The benefits of RECC have been illustrated by integrating this framework into a university system. In this system we have developed a scenario where intelligent services are implemented by combining different kind of knowledge such as the current context, user’s preferences and desired behaviors of the system. Such preferences and behaviors are expressed by means of rules in this scenario. The entire cycle of managing these rules, including the inference process and conflict detection, has been demonstrated in this scenario by means of the usage of RECC.

REFERENCES