Detecting Interactions between Behavioral Requirements with OWL and SWRL

Haibo Hu*, Dan Yang, Chunxiao Ye, Chunlei Fu And Ren Li

Abstract—High quality requirements analysis is one of the most crucial activities to ensure the success of a software project, so that requirements verification for software system becomes more and more important in Requirements Engineering (RE) and it is one of the most helpful strategies for improving the quality of software system. Related works show that requirement elicitation and analysis can be facilitated by ontological approaches and semantic web technologies. In this paper, we proposed a hybrid method which aims to verify requirements with structural and formal semantics to detect interactions. The proposed method is twofold: one is for modeling requirements with the semantic web language OWL, to construct a semantic context; the other is a set of interaction detection rules which are derived from scenario-based analysis and represented with semantic web rule language (SWRL). SWRL based rules are working with rule engines like Jess to reason in semantic context for requirements thus to detect interactions. The benefits of the proposed method lie in three aspects: the method (i) provides systematic steps for modeling requirements with an ontological approach, (ii) offers synergy of requirements elicitation and domain engineering for knowledge sharing, and (iii) the proposed rules can systematically assist in requirements interaction detection.

Keywords—Requirements Engineering, Semantic Web, OWL, Requirements Interaction Detection, SWRL.

I. INTRODUCTION

THE high quality of requirements analysis is one of the most crucial factors to ensure the success of a software project, so that requirements verification for complex software system becomes more and more important in requirements engineering and it is one of the most helpful strategies for improving the quality of software system.

One of the key issues for obtaining dependable requirements is to introduce actions as managing negative relationships among sets of requirements. Requirement Interaction is a negative and ubiquitous problem at the RE phase, that multiple requirements can’t be satisfied simultaneously when being integrated together, thus would cause unintended subsequences such as violation of functions or even introduce severe defects to a software system [1]-[2]. Moreover, with the introduction of different kinds of paradigms to software development, such as Component-Based Software Engineering (CBSE) [3], Feature-Oriented Software Development (FOSD) [4], Aspect-Oriented Programming (AOP) [5] and Service-Oriented Architecture (SOA) [6], the hazards of interactions can be more serious than simple failures in a single component, feature, aspect or service.

Related works show that requirements often interact with each other because of heterogeneity and diversity of stakeholders [7], inconsistency of requirements in semantics [8], contradictory decision of analysis & design [9], sharing of system context [10], etc. Hence, there is an obligatory need to managing requirement interactions that would answer questions such as: when and why do two requirements interact? How to detect interactions? And how do we resolve interaction?

In the past decade, some efforts have been paid on requirement interactions research both in academia and industry. These studies provide systematic approaches to requirements interaction taxonomies [11], modeling interactions with requirement models to figure out the nature of interactions [12], detecting interactions [13]-[15] or clear up interactions when integrating requirements [16]. Meanwhile, utilizing semantic web technologies is emerging in the field of software engineering, such as Ontology, RDF/OWL, Description Logics, Rule Markup Languages, etc., with their objectives of knowledge representation, sharing, and discovery by reasoning. To the best of our knowledge, no much work has been done for utilizing OWL/DL and SWRL based semantic web technologies to model and detect requirement interactions. Even though Chi-Lun Liu proposes an ontological method on requirements conflicts with analysis on activity diagram [17], his work does not include in depth on how to detect dynamic requirement interactions with semantic web technologies. On the other hand, existing achievements do not concern the synergy of requirements elicitation and domain engineering for knowledge sharing, or do they lack tools support for automation of interaction detection.

This paper proposes a hybrid method which aims to detect behavioral requirement interactions in order to improve the quality of software system. The proposed method includes two main parts. The first part of this method is an ontological modeling process for requirement specification thus to construct a semantic web based context. The second part of this
method is a set of conflicts detection rules derived from behavioral analysis and represented them into SWRL, which can be used to detect requirement interactions in former context with tool support like Protégé and JESS. The roadmap of proposed method in this paper is shown in Fig. 1.

The advantages of our proposed approach are threefold: On the one hand this method offers a systematic and step-by-step process for modeling ontologies and requirements in activity diagrams as well as analyzing conflicts. This method also offers a set of explicit questions for facilitating requirements elicitation works. Finally, requirement conflicts can be systematically detected by the proposed rules based on ontologies and requirement models.

The remainder of this paper is structured as follows: Section II gives an overview of the state of semantic web technologies. Section III presents the proposed process of requirement conflicts analysis based on ontologies. Section IV illustrates semantic based rules for conflicts detection and Section V provides the scenarios for demonstrating how these rules work appropriately. Finally, we conclude the paper in Section VI.

II. SEMANTIC WEB TECHNOLOGIES: OWL AND SWRL

The Semantic Web refers to both a vision and a set of technologies. The vision was first articulated by Tim Berners-Lee as an extension to the existing web in which knowledge and data could be published in a form easy for computers to understand and reason with. Doing so would support more sophisticated software systems that share knowledge, information and data on the Web just as people do by publishing text and multimedia. Under the stewardship of the W3C, a set of languages, protocols and technologies have been developed to partially realize this vision, to enable exploration and experimentation and to support the evolution of the concepts and technology.

The current set of W3C standards are based on RDF [18], a language that provides a basic capability of specifying graphs with a simple interpretation and serializing them in XML. Since it is a graph-based representation, RDF data are often reduced to a set of triples where each represents an edge in the graph (ClassA has-Relationship ClassB) or alternatively, a binary predication has-Relationship(ClassA, ClassB).

The Web Ontology Language OWL [19] is a family of knowledge representation languages based on Description Logics (DL) [20] with a representation in RDF. OWL supports the specification and use of ontologies that consist of terms representing individuals, classes of individuals, properties, and axioms that assert constraints over them. The axioms can be realized as simple assertions (e.g., Woman is a sub-class of Person, hasMother is a property from Person to Woman, Woman and Man are disjoint) and also as simple rules.

The use of OWL to represent requirements has several very important advantages that become critical in distributed environments involving coordination across multiple organizations. First, most policy languages define constraints over classes of targets, objects, actions and other constraints (e.g., time or location). A substantial part of the development of a policy is often devoted to the precise specification of these classes, e.g., the definition of what counts as a full time student or a public printer. This is especially important if the policy is shared between multiple organizations that must adhere to or enforce the policy even though they have their own native schemas or data models for the domain in question. The second advantage is that OWL’s grounding in logic facilitates the translation of policies expressed in OWL to other formalisms, either for analysis or for execution.

Semantic Web Rule Language (SWRL) [21] is a proposal for a Semantic Web rules-language, combining sublanguages of the OWL Web Ontology Language with those of the Rule Markup Language (RuleML) [22]. Rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

III. ONTOLOGICAL MODELING OF BEHAVIORAL REQUIREMENT

The output of the requirements engineering phase is a requirement artifacts that contain a set of requirements describing stakeholders’ needs. This set of requirements can either describe certain properties that have to be preserved (static view) or dynamic behavior which the system exhibits when certain triggers occur (dynamic view). Usually there is also a description of resources the system uses (environmental view).

In this work, we mainly take behavioral (or dynamic behavior) requirement into account for modeling and detection interactions. This is mainly because behavioral requirements are hold more properties in system context that are more likely to interact with each other.

A. Ontological Analysis of Behavioral Requirement

Our goal is to define OWL ontologies that can be used to represent the model of behavioral requirement and to show how they can be used to represent a context for interaction detection.

Each dynamic behavior requirement describes how the system should behave when it is in a certain state and a specific
trigger event occurs. For example, a requirement from the smart home system might state the following: “When the smoke sensor is triggered, smart home system will shut ovens, unlock doors and windows, and turn on fans”.

Any dynamic behavior requirement consists of internal attributes such as ID and Description, plus the following basic external attributes [11]:

- Prestate, which is a description of the required system state prior to the execution of this dynamic requirement;
- Trigger event, which is a description of the trigger event required for this dynamic requirement to execute;
- Action, which is a description of the action carried out by this dynamic requirement once triggered; and
- Next state, which describes the next state that the system reaches after executing this requirement.

Thus the objective properties of Behavioral Requirement can be extracted ontologically, namely, TriggerEvent, Action, Prestate and Nextstate. Moreover, Prestate and Nextstate are subclasses of Systemstate. We define objective properties in the ontology as hasTriggerEvent, hasAction, hasPrestate and hasNextstate, together with two data type properties as hasID and hasDescription. The behavioral requirements can be defined with OWL as follow:

```xml
<rdf:RDF
  xmlns:owl="http://www.w3.org/2002/07/owl"
  vocab:="http://www.example.com/">
  <owl:Ontology rdf:about="">
    <owl:Class rdf:ID="Systemstate"/>
    <owl:Class rdf:ID="Action"/>
    <owl:Class rdf:ID="Behavioral_Requirement"/>
    <owl:Class rdf:ID="TriggerEvent"/>
    <owl:Class rdf:ID="Nextstate"/>
    <owl:ObjectProperty rdf:ID="hasTriggerEvent"/>
    <owl:ObjectProperty rdf:ID="hasAction"/>
    <owl:ObjectProperty rdf:ID="hasPrestate"/>
    <owl:ObjectProperty rdf:ID="hasNextstate"/>
    <owl:DatatypeProperty rdf:ID="hasDescription"/>
    </owl:Ontology>
</rdf:RDF>
```

B. Refinement of Behavioral Requirement Model

In order to represent the relationships of requirements and system context, we propose an ontology design pattern that defines TriggerEvent, Action and SystemState by introducing three pairs of primitive concepts as (Trigger, Event), (Actuator, Actuation), (StateThing, StateValue). Primitive Classes such as Trigger, Actuator and Statething are collections of the entities come from a concrete problem domain, while Event, Actuation and Statevalue are predefined variables to indicate system behaviors or states.

Thus the behavioral requirement can be expressed with UML activity diagram, as shown in Fig. 2.

![Fig. 2 Modeling Behavioral Requirements with UML](image)

The refined model can be represented with OWL-DL by introducing three pairs of primitive concepts talked above, in addition with three sets of objective properties (hasTrigger, hasEvent), (hasActuator, hasActuation) and (hasStateThing, hasPrestateValue, hasNextstatevalue). We assert that Trigger and Actuator are system things in a domain, so that they are subclasses of StateThing.

1) Refinement for TriggerEvent

A TriggerEvent is a class that has exactly one trigger, which is an instance of the Trigger class, and one event which is an instance of the Event class.

```xml
TriggerEvent a rdf:s:Class.
  Trigger rdf:s:subClassOf StateThing.
  Trigger rdf:s:Property, owl:FunctionalProperty;
    rdf:s:domain TriggerEvent;
    rdf:s:range Trigger.
  Event a rdf:s:Property, owl:FunctionalProperty;
    rdf:s:domain TriggerEvent;
    rdf:s:range Event.
```

2) Refinement for Action

An Action is a class that has exactly one actuator, which is an instance of the Actuator class, and one event which is an
instance of the Event class.

Action a rdfs:Class.
  Actuator rdfs:subClassOf StateThing.
  Actuator rdfs:Property, owl:FunctionalProperty;
    rdfs:domain Action;
    rdfs:range Actuator.
  Actuation a rdfs:Property, owl:FunctionalProperty;
    rdfs:domain Action;
    rdfs:range Actuation.

3) Refinement for SystemState

A SystemState is a class that has exactly one statething, which is an instance of the StateThing class, one prestatevalue which is an instance of the StateValue class, and one nextstatevalue which is an instance of the StateValue class.

SystemState a rdfs:Class.
  StateThing a rdfs:Property, owl:FunctionalProperty;
    rdfs:domain SystemState;
    rdfs:range StateThing.
  StateValue a rdfs:class.
  PreStateValue rdfs:subClassOf StateValue.
  PreStateValue rdfs:Property, owl:FunctionalProperty;
    rdfs:domain SystemState;
    rdfs:range StateValue.
  NextStateValue rdfs:subClassOf StateValue.
  NextStateValue rdfs:Property, owl:FunctionalProperty;
    rdfs:domain SystemState;
    rdfs:range StateValue.

C. Behavioral Requirement Ontology

Based on above analysis and refinement, the Ontology of behavioral requirement can be expressed with a UML class diagram, as shown in Fig. 3.

![UML class diagram](image)

Fig. 2 Modeling Behavioral Requirements with UML

Alternatively, Event, Actuation and Statevalue can be defined in OWL as datatype properties of classes

TriggerEvent, Action and SystemState respectively. In this paper, we only take them as objective properties for scalability. Although the cost of inference will be increase, we can introduce owl:sameAs and owl:differentFrom to reason on classes for feasibility and decidability of rules. This will be mentioned in Section V.

IV. MODELING FOR REQUIREMENTS INTERACTION

A. Formal Description of Behavioral Requirement

A specification \( \theta \) for a behavioral requirement is a triplet \( \theta = (T, A, S) \) that:

\[
T, A \vdash S
\]

where,

- \( T \) is the set of trigger events with \( m \) elements \( t \), and \( t \) is a pair \( t=(r, e) \) representing trigger and event;
- \( A \) is the set of actions with \( n \) elements \( a \), and \( a \) is a pair \( a=(o, i) \) representing actuator and actuation;
- \( S \) is the set of system-state with \( k \) elements \( s \), and \( s \) is a triplet \( s=(g, p, n) \) representing state-thing, values of pre-state and next-state; i.e.,

\[
\forall r_j \in T, a_j \in A, s_j \in S, \land_{j}(r_j, e_j), \land_{j}(o_j, i_j) = \land_{j}(g, p, n)
\]

B. Interaction between Behavioral Requirements

Two aspects of the challengeable issue for requirements interaction are that how do they interact with each other, and when will they interact. The former concerns about the classification of feature interaction, while the latter leads to work on feature interaction detection.

With our ontological modeling, behavioral requirement interaction is defined as an object property among requirements, i.e., it holds binary relation of both its Domain and Range as the class Behavioral_Requirements. In our work, we summarize 6 types of requirement interactions from Shehata et al’s taxonomy [11], i.e., Non-Determinism, Override, Dependence, Negatively Impact, Bypass, and Infinite Loop, which are specified as sub-properties of Interaction in our Ontology.

The interactions are defined as properties in our ontology shown in Table I.

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Denotation</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Determinism</td>
<td>non_Determine</td>
<td>Requirement</td>
<td>Requirement</td>
</tr>
<tr>
<td>Override</td>
<td>override</td>
<td>Requirement</td>
<td>Requirement</td>
</tr>
<tr>
<td>Dependence</td>
<td>depends_On</td>
<td>Requirement</td>
<td>Requirement</td>
</tr>
<tr>
<td>Negatively Impact</td>
<td>negatively_Impact</td>
<td>Requirement</td>
<td>Requirement</td>
</tr>
<tr>
<td>Bypass</td>
<td>byPass</td>
<td>Requirement</td>
<td>Requirement</td>
</tr>
<tr>
<td>Infinite Loop</td>
<td>infinite_Loop</td>
<td>Requirement</td>
<td>Requirement</td>
</tr>
</tbody>
</table>

TABLE I

INTERACTIONS AS PROPERTIES OF REQUIREMENT
A common characteristic of requirement interactions is that they are subtle in nature and resulting from sharing of system context. Furthermore, requirements usually interact with each other in different manners under specific scenarios of sharing system context. Therefore, analysis on system context sharing is the hinge for interaction detection.

For the conditions of system context variables comparison, we revised the work by Shehata et al [11] of 9 scenarios of dynamic behavioral requirement to 7 scenarios. Each scenario is constructed with a set of element conditions and a decision. The element conditions for deciding requirement interactions are Same Trigger-events, Linked Trigger-events, Dual linked Trigger-events, Same Pre-states, Different Next-states, Overridden Actions, Negatively Impacted Actions, Dependence of Actions, Action Bypass on Pre-state, and Order.

The system context of a behavioral requirement is presented with semantic data by classes and their properties. The scenario for system context sharing is analyzed by comparing variables of a pair of behavioral requirements. In order to facilitate interaction detections with semantic rules, we investigate the variables for a pair of behavioral requirements, and make detailed comparison to map each element condition to OWL

### TABLE II

<table>
<thead>
<tr>
<th>Element Condition</th>
<th>Denotation Rule Expression</th>
</tr>
</thead>
</table>
| Sharing System Context | saCo(?u,?u) ← saCo(?u,?u) ← \exists g \exists g |}
| Same Trigger-events | saTe(?u,?u) ← saTe(?u,?u) ← \exists g \exists g |}
| Linked Trigger-events | liTe(?u,?u) ← liTe(?u,?u) ← \exists g \exists g |}
| Dual-linked Trigger-events | dlTe(?u,?u) ← dlTe(?u,?u) ← \exists g \exists g |}
| Same Pre-states | saPs(?u,?u) ← saPs(?u,?u) ← \exists g \exists g |}
| Different Next-states | diNn(?u,?u) ← diNn(?u,?u) ← \exists g \exists g |}
| Overridden Actions | ovAc(?u,?u) ← ovAc(?u,?u) ← \exists g \exists g |}
| Dependence of Actions | dec(?u,?u) ← dec(?u,?u) ← \exists g \exists g |}
| Negatively Impacted Actions | niAc(?u,?u) ← niAc(?u,?u) ← \exists g \exists g |}
| Create Action | crAc(?u,?u) ← crAc(?u,?u) ← \exists g \exists g |}

### C. Feature Interaction Detection Guidelines

With analysis for each scenario of behavioral requirement interactions, interaction detection guidelines can be presented by calling for above conditions of sharing context, as shown in Table III.

### TABLE III

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Formal Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-Determinism</td>
<td>saTe(?u,?u) ∧ saPs(?u,?u) ∧ diNn(?u,?u) ← non_Determine(?u,?u)</td>
</tr>
<tr>
<td>Dependence</td>
<td>saTe(?u,?u) ∧ saPs(?u,?u) ∧ dec(?u,?u) ← deps_Ont(?u,?u)</td>
</tr>
<tr>
<td>Override</td>
<td>liTe(?u,?u) ∧ ovAc(?u,?u) ← override(?u,?u)</td>
</tr>
<tr>
<td>Negative Impact</td>
<td>liTe(?u,?u) ∧ niAc(?u,?u) ← negatively_Impact(?u,?u)</td>
</tr>
<tr>
<td>Bypass</td>
<td>liTe(?u,?u) ∧ bpAc(?u,?u) ← bypass(?u,?u)</td>
</tr>
<tr>
<td>Infinite Loop</td>
<td>dlTe(?u,?u) ∧ crAc(?u,?u) ∧ crAc(?u,?u) ← infinite_Loop (?u,?u)</td>
</tr>
</tbody>
</table>

### V. Interaction Detection with SWRL

#### A. Semantic Web Context for Interaction Detection

To detect interactions with the proposed method, a Semantic Web based context should be constructed to provide a knowledge base with precise and unambiguous specification of requirements in problem domain. The mission is performed in four steps with the aids of Semantic Web tools Protégé 3.4.4.

Step I: Merge the conceptual model of behavioral requirement to the problem domain ontology by defining classes, properties with their domain, range and characteristics.

Step II: Extract statething, trigger and actuators from the textual specification of requirement with the conceptual model, and precisely identify their values with no ambiguity.

Step III: Add the outcome individuals of step2 to the behavioral requirement ontology, and refine each specification as a distinctive individual of class Behavioral Requirement.

Step IV: To ensure the vectors comparable, individuals of actuation, event and system should be well defined and related. This task can be done by referring owl:sameAs and owl:differentFrom assertions to indicate that they actually refer to the same thing or not.

Thus the system context with a set of classes and proper ties represented with semantic data can infer assertion of properties by applying SWRL rules.

#### B. Detecting Interactions by Reasoning with SWRL

In our work, the SWRL is adopted as a rule language and Jess as Rule Engine for reasoning to detect requirement interactions in the semantic web based context. The rules are expressed with SWRL syntax by importing semantics derived from formalized interaction detecting conditions and guidelines in Table II and III.

Due to the limitation of space, for element condition rules, we only take the SWRL rule of element conditions for same-TriggerEvents as example, which is presented as below.

#### Rule- same-TriggerEvents:

```
Feature(?fa) ∧ Feature(?fb) ∧ differentFrom(?fa,?fb) ∧
hasTriggerEvent(?fa,?fa) ∧ hasTriggerEvent(?fa,?fa) ∧
hasTriggerEvent(?fa,?fa) ∧ hasTriggerEvent(?fa,?fa) ∧
hasEvent(?fa,?fa) ∧ hasEvent(?fa,?fa) ∧
sameAs(?fa,?fa) sameAs(?fa,?fa) sameAs(?fa,?fa) → saTe(?fa,?fa)
```
The interaction detection guideline for Non-Determinism, Dependence, Override, Negative Impact, Bypass and Infinite Loop can be represented with SWRL rule as follows,

Rule1- Interaction Detection: Non-Determinism:

\[ \text{Feature}(\text{?fa}) \land \text{Feature}(\text{?fb}) \land \\
\text{saTe}(\text{?fa}, \text{?fb}) \land \text{saPs}(\text{?fa}, \text{?fb}) \land \text{diNs}(\text{?fa}, \text{?fb}) \land \\
\text{differentFrom}(\text{?fa}, \text{?fb}) \rightarrow \text{non\_Determine}(\text{?fa}, \text{?fb}) \]

Rule2- Interaction Detection- Dependence:

\[ \text{User\_Policy}(\text{?ua}) \land \text{User\_Policy}(\text{?ub}) \land \\
\text{saTe}(\text{?ua}, \text{?ub}) \land \text{saPs}(\text{?ua}, \text{?ub}) \land \text{deAc}(\text{?ua}, \text{?ub}) \land \\
\text{differentFrom}(\text{?ua}, \text{?ub}) \rightarrow \text{overide}(\text{?ua}, \text{?ub}) \]

Rule3- Interaction Detection- Override (a):

\[ \text{User\_Policy}(\text{?ua}) \land \text{User\_Policy}(\text{?ub}) \land \\
\text{saTe}(\text{?ua}, \text{?ub}) \land \text{ovAc}(\text{?ua}, \text{?ub}) \land \\
\text{differentFrom}(\text{?ua}, \text{?ub}) \rightarrow \text{override}(\text{?ua}, \text{?ub}) \]

Rule3- Interaction Detection- Override (b):

\[ \text{User\_Policy}(\text{?ua}) \land \text{User\_Policy}(\text{?ub}) \land \\
\text{liTe}(\text{?ua}, \text{?ub}) \land \text{ovAc}(\text{?ua}, \text{?ub}) \land \\
\text{differentFrom}(\text{?ua}, \text{?ub}) \rightarrow \text{override}(\text{?ua}, \text{?ub}) \]

Rule4- Interaction Detection- Negative Impact (a)

\[ \text{User\_Policy}(\text{?ua}) \land \text{User\_Policy}(\text{?ub}) \land \\
\text{saTe}(\text{?ua}, \text{?ub}) \land \text{niAc}(\text{?ua}, \text{?ub}) \land \\
\text{differentFrom}(\text{?ua}, \text{?ub}) \rightarrow \text{negatively\_Impact}(\text{?ua}, \text{?ub}) \]

Rule4- Interaction Detection- Negative Impact (b)

\[ \text{User\_Policy}(\text{?ua}) \land \text{User\_Policy}(\text{?ub}) \land \\
\text{liTe}(\text{?ua}, \text{?ub}) \land \text{niAc}(\text{?ua}, \text{?ub}) \land \\
\text{differentFrom}(\text{?ua}, \text{?ub}) \rightarrow \text{negatively\_Impact}(\text{?ua}, \text{?ub}) \]

Rule5- Interaction Detection- Bypass

\[ \text{User\_Policy}(\text{?ua}) \land \text{User\_Policy}(\text{?ub}) \land \\
\text{liTe}(\text{?ua}, \text{?ub}) \land \text{hpAp}(\text{?ua}, \text{?ub}) \land \\
\text{differentFrom}(\text{?ua}, \text{?ub}) \rightarrow \text{byPass}(\text{?ua}, \text{?ub}) \]

Rule6- Interaction Detection- Infinite Loop

\[ \text{User\_Policy}(\text{?ua}) \land \text{User\_Policy}(\text{?ub}) \land \\
\text{diTe}(\text{?ua}, \text{?ub}) \land \text{crAc}(\text{?ua}, \text{?ub}) \land \text{crAc}(\text{?ub}, \text{?ua}) \land \\
\text{differentFrom}(\text{?ua}, \text{?ub}) \rightarrow \text{infinite\_Loop}(\text{?ua}, \text{?ub}) \]

**C. Tools Support for Interaction Detection**

In our work, Protégé [23] v3.4.4 is employed as ontology editor with plug-in SWRLTab for editing rules that can be bridged to rule engine such as Jess [24]. With these tools support, requirements can be refined and specified into OWL, thus to represent the semantic context for requirement interaction detection. A snapshot of the tools used in our interaction detection work is shown as Fig. 3.

**D. Experimental Study and Discussions**

With the proposed method in this paper, we take the experimental case provided in [25] for validation. The case is derived from smart home system as the problem domain, which is attracted attention due to feature interactions in both requirement and networks.

Core concepts of the smart home system domain are mapped to behavioral requirement ontology with above structural analysis and refinement. There are 20 classes from problem domain and 38 individuals of behavioral requirement in OWL-based semantic web context. The detection result of behavioral interactions is shown in Fig. 4 as follow.

![Fig. 3 Snapshot of Protégé and SWRLTab](image3)

![Fig. 4 Results of Experiment in Smart Home System](image4)

It can be seen from the graph in Fig. 4 that 91 possible interactions are inferred from rules with the proposed method. With comparison to IRIS method in [25], the proposed method in this paper has threefold benefit: the method (i) provides systematic steps for modeling requirements with an ontological approach, (ii) offers synergy of requirements elicitation and domain engineering for knowledge sharing, and (iii) the proposed rules can systematically assist in requirements
interaction detection with tools support for automation. In addition, the proposed method is a formal approach for interaction detection that based on semantic web technologies such as OWL and SWRL.

While the current limitations of the proposed method in our work should be figured out before we draw a conclusion. For one thing, the complexity and diversity of requirement interaction problem results in multiple aspects of manifestation. We cannot assert that the proposed method works on all kinds of interactions effectively. It should be taken as a complementary methodology to current means for requirement interaction detection. On the other hand, current Ontological method and Semantic Web technologies are not good at representing dynamical and temporal relations in semantics and the price on decidability of rules and logical inference should also be taken into account. Nevertheless, the proposed method will catch its advantages to overcome its disadvantages by shaping itself on the specific problem domain and introducing more powerful emerging technologies from the Semantic Web community. In our feature work, the detailed mechanism of interaction will be further investigated, and how to avoid interactions when integrating requirements will also be taken into account.

VI. CONCLUSION

This paper proposes a hybrid method which aims to detect behavioral requirement interactions in order to improve the quality of software system. The proposed method includes two main parts. The first part of this method is an ontological modeling process for requirement specification thus to construct a semantic web based context. The second part of this method is a set of conflicts detection rules derived from behavioral analysis and represented with SWRL, which can be used to detect requirement interactions with tools support for automation. The case experiment study results show that the proposed method can detection requirement interactions upon a semantic web based system context beneficially.

ACKNOWLEDGMENT

This work is supported by the National S&T Major Project of China under grant No. 2009ZX07315-006, NSF Foundation of China under Grant No. 60803027, and the Natural Science Foundation in Chongqing City of China under Grant No. 2008BB2312.

REFERENCES


