International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:12, 2010

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 (3)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 stake-holders [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

Haibo HU is lecturer with the School of Software Engineering, D-Campus Chongqing University, Chongqing, 401331, China (*Corresponding author, phone:+862365127222; fax:+862365678333; e-mail: hbhu@cqu.edu.cn).
Dan YANG is with the School of Software Engineering, Chongqing University, Chongqing, 401331, China (e-mail: danyang@cqu.edu.cn).
Chunxiao YE is with the College of Computer Science, Chongqing University, Chongqing, 400030, China (e-mail: cxye@cqu.edu.cn).
Chunlei FU is with the Centre for Information and Network of Chongqing University, Chongqing, 400030, China (e-mail: clfu@cqu.edu.cn).
Ren LI is PhD student with the College of Computer Science, Chongqing University, Chongqing, 400030, China (e-mail: renlee@cqu.edu.cn).

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.

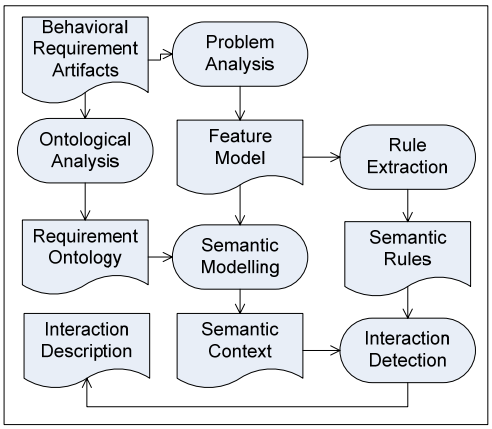

Fig. 1 Framework of Proposed Method

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 onotologies 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 sub-class 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:

```
<rdf:RDF
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Systemstate"/>
  <owl:Class rdf:ID="Prestate">
    <rdfs:subClassOf rdf:resource="#Systemstate"/>
  </owl:Class>
  <owl:Class rdf:ID="Action"/>
  <owl:Class rdf:ID="Behavioral_Requirement"/>
  <owl:Class rdf:ID="Triggerevent"/>
  <owl:Class rdf:ID="Nextstate">
    <rdfs:subClassOf rdf:resource="#Systemstate"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="hasTriggerevent">
    <rdfs:range rdf:resource="#Action"/>
    <rdfs:domain rdf:resource="#Behavioral_Requirement"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasAction">
    <rdfs:range rdf:resource="#Action"/>
    <rdfs:domain rdf:resource="#Behavioral_Requirement"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasPrestate">
    <rdfs:range rdf:resource="#Prestate"/>
    <rdfs:domain rdf:resource="#Behavioral_Requirement"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasNextstate">
    <rdfs:domain rdf:resource="#Behavioral_Requirement"/>
    <rdfs:range rdf:resource="#Nextstate"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="hasDescription">
    <rdfs:domain rdf:resource="#Behavioral_Requirement"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="hasID">
    <rdfs:domain rdf:resource="#Behavioral_Requirement"/>
    <rdf:type rdf:resource
        ="http://www.w3.org/2002/07/owl#Dataty peProperty"/>
    </owl:FunctionalProperty>
</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 SytemState 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.

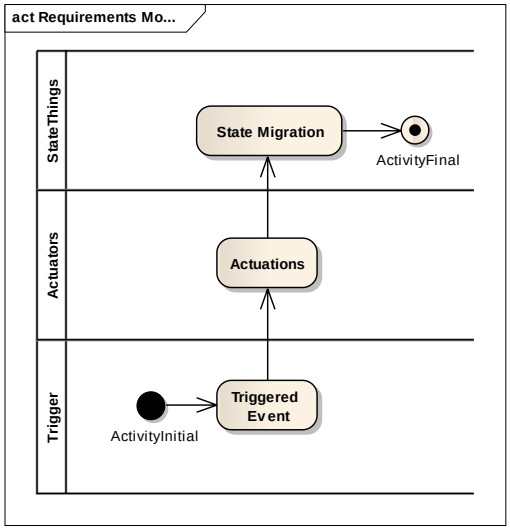

Fig. 2 Modeling Behavioral Requirements with UML

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.

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

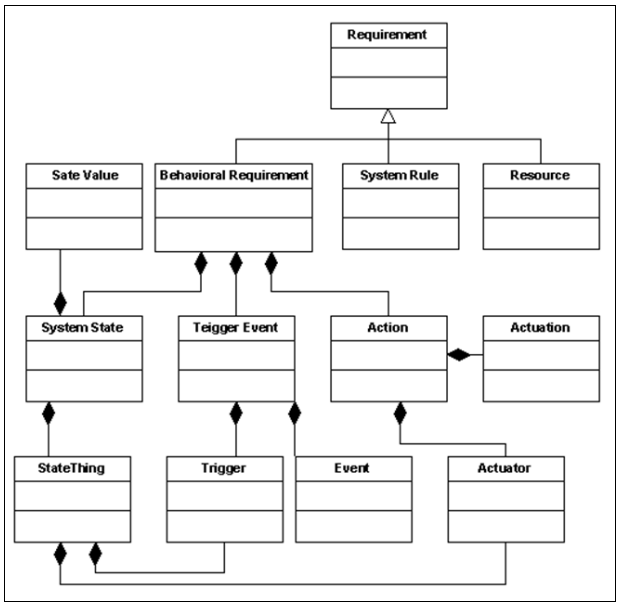

Fig. 2 Modeling Behavioral Requirements with UML

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 θ for a behavioral requirement is a triplet θ = (*T*, *A*, *S*) that:

$$T, A \models S \tag{1}$$

where,

T is the set of trigger events with *m* elements *t*, and *t* is a pair *t*=(*r, e*) representing t**r**igger and **e**vent;

A is the set of **a**ctions with *n* elements *a*, and *a* is a pair *a*=(*o*, *i*) representing actuat**o**r and actuat**i**on;

S is the set of system-state with *k* elements *s*, and *s* is a triplet s= (*g*, *p*, *n*) representing state-thin**g**, values of **p**re-state and **n**ext-state; i.e.,

$$\forall r_j \in T, a_j \in A, s_j \in S, \bigwedge_{j=1}^{m}(r_j, e_j), \bigwedge_{j=1}^{n}(o_j, i_j) \models \bigwedge_{j=1}^{k}(g_j, p_j, n_j) \tag{2}$$

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 I
INTERACTIONS AS PROPERTIES OF REQUIREMENT

Interaction	Denotation	Domain	Range
Interaction	*interact_With*	Requirement	Requirement
Non-Determinism	*non_Determine*	Requirement	Requirement
Override	*override*	Requirement	Requirement
Dependence	*depends_On*	Requirement	Requirement
Negatively Impact	*negatively_Impact*	Requirement	Requirement
Bypass	*byPass*	Requirement	Requirement
Infinite Loop	*infinite_Loop*	Requirement	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, Overrided 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 based model. The element conditions of behavioral requirement interaction are summarized with formal description with their logical characteristics, as shown in Table II.

TABLE II
ELEMENT CONDITIONS FOR INTERACTION DETECTION

Element Condition	Denotation	Rule Expression
Sharing System Context	$shCo(u_1,u_2)$	$shCo(u_1,u_2) \leftarrow \exists\, g_j=g_k$
Same Trigger-events	$saTe(u_1,u_2)$	$saTe(u_1,u_2) \leftarrow \exists\, (r_j,e_j)=(r_k,e_k)$
Linked Trigger-events	$liTe(u_1,u_2)$	$liTe(u_1,u_2) \leftarrow \exists\, g_j=g_k \wedge\ p_i=n_j$
Duallinked Triggerevents	$dlTe(u_1,u_2)$	$dlTe(u_1,u_2) \leftarrow liTe(u_1,u_2) \wedge\ liTe(u_2,u_1)$
Same Pre-states	$saPs(u_1,u_2)$	$saPs(u_1,u_2) \leftarrow \exists\, (g_j,p_j)=(g_k,p_k)$
Different Next-states	$diNs(u_1,u_2)$	$diNs(u_1,u_2) \leftarrow \exists\, g_j=g_k \wedge\ n_j \neq n_k$
Override Actions	$ovAc(u_1,u_2)$	$ovAc(u_1,u_2) \leftarrow \exists\, g_j=g_k \wedge\ p_k=n_k \neq n_j$
Dependence of Actions	$deAc(u_1,u_2)$	$deAc(u_1,u_2) \leftarrow \exists\, g_j=o_k \wedge\ i_k=p_j=n_j$
Negative Impact Actions	$niAc(u_1,u_2)$	$niAc(u_1,u_2) \leftarrow \exists\, g_j=g_k \wedge\ (g_j,n_j) \neq (g_k,n_k)$
ActionBypass on Prestate	$bpAp(u_1,u_2)$	$bpAp(u_1,u_2) \leftarrow \exists\, g_j=g_k \wedge\ (g_j,n_j) \neq (g_k,p_k)$
Create Action	$crAc(u_1,u_2)$	$crAc(u_1,u_2) \leftarrow \exists\, (g_j,n_j)=(r_k,e_k) \wedge\ n_j \neq p_j$

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
FORMALIZED INTERACTION DETECTION GUIDELINES

Interaction	Formal Expressions
non-Determinism	$saTe(u_1,u_2) \wedge\ saPs(u_1,u_2) \wedge\ diNs(u_1,u_2) \rightarrow non\_Determine(u_1,u_2)$
Dependence	$saTe(u_1,u_2) \wedge\ saPs(u_1,u_2) \wedge\ deAc(u_1,u_2) \rightarrow depens\_On(u_1,u_2)$
Override	$liTe(u_1,u_2) \wedge\ ovAc(u_1,u_2) \rightarrow override(u_1,u_2)$
Negative Impact	$liTe(u_1,u_2) \wedge\ niAc(u_1,u_2) \rightarrow negatively\_Impact(u_1,u_2)$
Bypass	$liTe(u_1,u_2) \wedge\ bpAp(u_1,u_2) \rightarrow byPass(u_1,u_2)$
Infinite Loop	$dlTe(u_1,u_2) \wedge\ crAc(u_1,u_2) \wedge\ crAc(u_2,u_1) \rightarrow infinite\_Loop(u_1,u_2)$

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 is able to 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:

$Feature(?fa) \wedge\ Feature(?fb) \wedge\ differentFrom(?fa,?fb) \wedge$
$hasTriggerEvent(?fa,?ta) \wedge\ hasTrigger(?ta,?ra) \wedge$
$hasTriggerEvent(?fb,?tb) \wedge\ hasTrigger(?tb,?rb) \wedge$
$hasEvent(?tb,?eb) \wedge\ hasEvent(?ta,?ea) \wedge$
$sameAs(?ra,?rb) \wedge\ sameAs(?ea,?eb) \rightarrow saTe(?fa,?fb)$

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:

Feature(?fa)∧ *Feature* (?fb)∧

saTe(?fa,?fb)∧ *saPs*(?fa,?fb)∧ *diNs*(?fa,?fb)∧

differentFrom(?fa,?fb)∧ →*non_Determine*(?fa,?fb)

Rule2- Interaction Detection- Dependence:

User_Policy(?ua)∧ *User_Policy*(?ub)∧

saTe(?ua,?ub)∧ *saPs*(?ua,?ub)∧ *deAc*(?ua,?ub)∧
differentFrom(?ua,?ub)→*non_Determine*(?ua,?ub)

Rule3- Interaction Detection- Override (a):

User_Policy(?ua)∧ *User_Policy*(?ub)∧

saTe(?ua,?ub)∧ *ovAc*(?ua,?ub)∧
differentFrom(?ua,?ub)→*override*(?ua,?ub)

Rule3- Interaction Detection- Override (b):

User_Policy(?ua)∧ *User_Policy*(?ub)∧

liTe(?ua,?ub)∧ *ovAc*(?ua,?ub)∧
differentFrom(?ua,?ub)→*override*(?ua,?ub)

Rule4- Interaction Detection- Negative Impact (a)

User_Policy(?ua)∧ *User_Policy*(?ub)∧

saTe(?ua,?ub)∧ *niAc*(?ua,?ub)∧
differentFrom(?ua,?ub)→*negatively_Impact*(?ua,?ub)

Rule4- Interaction Detection- Negative Impact (b)

User_Policy(?ua)∧ *User_Policy*(?ub)∧

liTe(?ua,?ub)∧ *niAc*(?ua,?ub)∧

differentFrom(?ua,?ub)∧ →*negatively_Impact*(?ua,?ub)

Rule5- Interaction Detection- Bypass

User_Policy(?ua)∧ *User_Policy*(?ub)∧

liTe(?ua,?ub)∧ *bpAp*(?ua,?ub)∧

differentFrom(?ua,?ub)∧ →*byPass*(?ua,?ub)

Rule6- Interaction Detection- Infinite Loop

User_Policy(?ua)∧ *User_Policy*(?ub)∧

dlTe(?ua,?ub)∧ *crAc*(?ua,?ub)∧ *crAc*(?ub,?ua)∧
differentFrom(?ua,?ub)→*infinite_Loop*(?ua,?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.

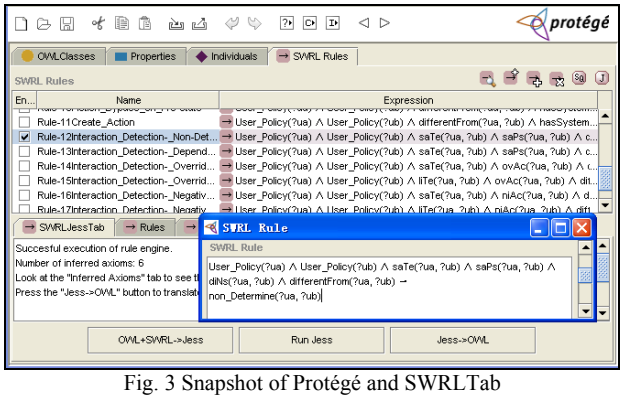

Fig. 3 Snapshot of Protégé and SWRLTab

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.

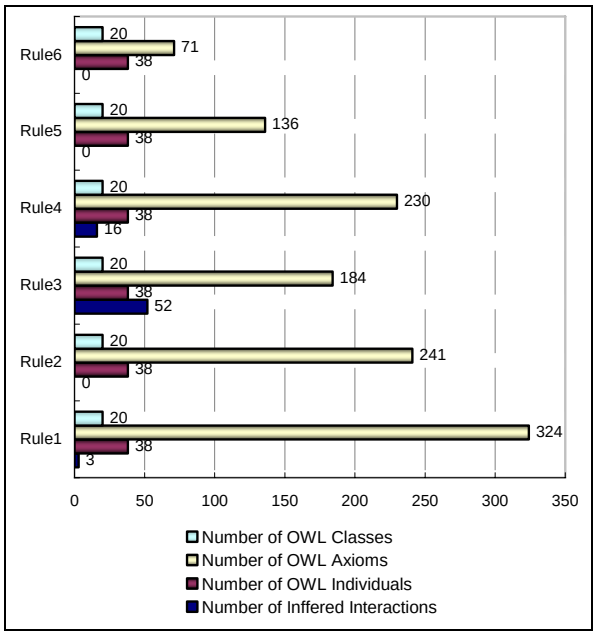

Fig. 4 Results of Experiment in Smart Home System

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 (3)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

[1] A. Nhlabatsi, R. Laney, and B. Nuseibeh, “Feature interaction: The security threat from within software systems,” Progress in Informatics, no. 5, pp. 75-90, 2008.

[2] R. G. Crespo, “Identification of feature denial of services,” in 2nd International Conference on Next Generation Mobile Applications, Services, and Technologies, NGMAST 2008, September 16-19, 2008, pp. 571-575, 2008.

[3] W. Pree, “Component-based software development - a new paradigm in software engineering?,” in Proceedings of the 1997 Asia-Pacific Software Engineering Conference and International Computer Science Conference, APSEC'97 and ICSC'97, December 2-5, 1997, pp. 523-524, 1997.

[4] S. Apel and C. Kästner, “An Overview of Feature-Oriented Software Development,” Journal of Object Technology, vol. 8, no. 5, pp. 49-84, Aug. 2009.

[5] T. Elrad, R. E. Filman, and A. Bader, “Aspect-oriented programming,” Communications of the ACM, vol. 44, no. 10, pp. 29-32, 2001.

[6] G. A. Lewis, E. Morris, S. Simanta, and L. Wrage, “Effects of Service-Oriented Architecture on software development lifecycle activities,” Software Process Improvement and Practice, vol. 13, no. 2, pp. 135-144, 2008.

[7] I. Sommerville, “Integrated requirements engineering: A tutorial,” IEEE Software, vol. 22, no. 1, pp. 16-23, 2005.

[8] V. Gervasi and D. Zowghi, “Reasoning about inconsistencies in natural language requirements,” ACM Transactions on Software Engineering and Methodology, vol. 14, no. 3, pp. 277-330, 2005.

[9] W. N. Robinson, S. D. Pawlowski, and V. Volkov, “Requirements Interaction Management,” ACM Computing Surveys, vol. 35, no. 2, pp. 132-190, Jun. 2003.

[10] A. Nhlabatsi, “Initialisation Problems in Feature Composition,” PhD thesis, The Open University, 2009.

[11] M. Shehata, A. Eberlein, and A. Fapojuwo, “A taxonomy for identifying requirement interactions in software systems,” Computer Networks, vol. 51, no. 2, pp. 398-425, Feb. 2007.

[12] A. Classen, P. Heymans, and P. Schobbens, “What's in a Feature: A Requirements Engineering Perspective,” in Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering, vol. 4961, pp. 16-30, 2008.

[13] M. Heisel and J. Souquières, “A Heuristic Approach to Detect Feature Interactions in Requirements,” in Feature Interactions in Telecommunications and Software Systems V, pp. 165-171, 1998.

[14] E. Sarmiento, M. R. S. Borges, and M. L. M. Campos, “Applying an event-based approach for detecting requirements interaction,” in ICEIS 2009 - 11th International Conference on Enterprise Information Systems, May 6-10, 2009, pp. 225-230, 2009.

[15] M. Shehata, “Detecting Requirements Interactions Using Semi-Formal Methods,” PhD thesis, University of Calgary, 2005.

[16] R. Laney, T. T. Tun, M. Jackson, and B. Nuseibeh, “Composing Features by Managing Inconsistent Requirements,” in Feature Interactions in Software and Communication Systems IX, pp. 129-144, 2007.

[17] C. Liu, “Ontology-based requirements conflicts analysis in activity diagrams,” in International Conference on Computational Science and Its Applications, ICCSA 2009, June 29-July 2, 2009, LNCS, vol. 5593, pp. 1-12, 2009.

[18] O. Lassila and R. R. Swick, “Resource Description Framework (RDF) Model and Syntax Specification.” [Online]. Available: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. [Accessed: 02-Oct-2010].

[19] S. Bechhofer et al., “OWL Web Ontology Language Reference.” [Online]. Available: http://www.w3.org/TR/owl-ref/. [Accessed: 07-Oct-2010].

[20] F. Baader et al. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003.

[21] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, “SWRL: A Semantic Web Rule Language Combining OWL and RuleML.” [Online]. Available: http://www.w3.org/Submission/SWRL/. [Accessed: 07-Oct-2010].

[22] H. Boley, S.Tabet, and G.Wagner, “Design rationale of RuleML: a markup language for Semantic Web rules”. Proc. Semantic Web Working Symp, California, USA, Aug. 2001, pp. 381-401, 2001.

[23] N. Noy, M. Sintek, S. Decker, M. Crubezy, R. Fergerson, and M. Musen, “Creating semantic web contents with protege-2000,” IEEE Intelligent Systems and Their Applications, vol. 16, no. 2, pp. 60-71, 2001.

[24] E. Wang and Y. S. Kim, “A teaching strategies engine using translation from SWRL to Jess,” in 8th International Conference on Intelligent Tutoring Systems, ITS 2006, June 26-30, 2006, LNCS vol. 4053, pp. 51-60, 2006.

[25] M. Shehata, A. Eberlein, and A. Fapojuwo, “Using semi-formal methods for detecting interactions among smart homes policies,” Science of Compter Programming, vol. 67, no. 2, pp. 125-161, Jul. 2007.