Handling Inconsistency in Software Requirements

Richa Sharma[1] and K. K. Biswas[2]
[1]*School of Information Technology, IIT Delhi, India*
[2]*Department of Computer Science, IIT Delhi, India*

Keywords: Requirements Specification, Inconsistency, Presupposition, Knowledge Representation, Courteous Logic.

Abstract: Software Requirements expressed in the form of natural language are often informal and possibly vague. The need for formal representation of the requirements has been explored and addressed in various forms earlier. Of several recommended approaches, logical representation of requirements has been widely acknowledged to formalize the requirements languages. In this paper, we present courteous logic based representations for software requirements. We report the benefits of courteous logic based representations for handling inconsistencies in software requirements and take into account views of multiple stakeholders and the presuppositions. We show how courteous logic based representations can be used to ensure consistency as well as to uncover presuppositions in the requirements.

1 INTRODUCTION

Software Development commences with the phase comprising requirement engineering activities. Any software model, whether it is Waterfall model, Iterative or Agile, commences with tasks centred on the popularly known requirements phase. The requirements phase becomes crucial to the success of the software as this phase only serves as the basis for subsequent phases of software development. It would not be incorrect to say that establishing correct requirements is imperative even in agile model where developers work in close connection with the users; there is still need to clearly specify the requirements so that the requirements are well understood by the developer. The mode of expressing the requirements may vary from visual models to textual use-cases to user scenarios, but the underlying idea is that the requirements should be a good representation of the domain under study and should be well understood and agreed upon by all the stakeholders. There is no precise definition as to what is meant by 'good' representation but with reference to features of good software requirements specification, requirements representations should be consistent, unambiguous and complete in nature (IEEE CS, 1998). It is a challenging task to draw upon the representations satisfying these qualities. The most common defect that arises in software requirements representation is that of *inconsistency*. The elicited requirements describing the functional specifications of the behaviour of the information system or the software application are frequently incomplete in nature as users might not be able to express all what they need or the stated requirements are not well understood by the requirements engineer. Therefore, the gathered requirements often need enrichment during analysis for functional behaviour as well as non-functional properties. Requirements engineers need to examine these gathered requirements and to transform this "rough" sketch of requirements into a correct requirement specification (Zowghi, 2003). As a result, new requirements are identified that should be added to the specification, or some of the previously stated requirements may need to be deleted to improve the specification. It becomes a critical task for requirements engineers to maintain the consistency of the set of specified requirements.

Our contribution in this paper is to demonstrate the courteous logic based requirements representations (requirements specification hereafter) as a worthwhile approach towards handling inconsistency in the requirements and the related issue of identifying presuppositions (Ma et al., 2009). The use of logic to formalize the requirements has been acknowledged in earlier works too as logic offers proof-theoretic framework facilitating requirements validation and evolution. We are making use of a form of non-monotonic logic as requirements evolution becomes non-monotonic after an initial monotonic phase (Zowghi, 2003).

The rest of the paper is organized as follows. In section 2, we elaborate the inconsistency concern that motivated the use of courteous logic. Section 3 describes courteous logic in detail and how it can be used to address the inconsistency and presupposition concern in requirements. In section 4, we present an overview of related work followed by possible threats to the usage of formal logic by requirements analysts in section 5. Section 6 finally presents discussion and conclusion.

2 INCONSISTENCY CONCERN

Several adequate definitions are available elucidating the inconsistency concern in software requirements (Tsai et al., 1992), (Gervasi and Zowghi, 2005). Of the two IEEE interpretations of inconsistency, namely internal inconsistency and the Software Requirements Specification not agreeing with some higher level document (traceability), we are interested in addressing the former one. Internal inconsistency arises when the specification of requirements contains conflicting or contradictory information about the expected behaviour of the system. The reason for conflicting information existence can be attributed to multiple views of the stakeholders. Internal inconsistency may also arise when some new requirement is added or an existing requirement is removed from the requirements specification. Internal inconsistency may also arise due to lack of domain knowledge at the requirements engineer's end owing to which some implicit knowledge, referred to as presupposition that could not find explicit expression in the requirements set, gets missed in the elicited requirements. These forms of inconsistency may be detected or remain undetected at the time of requirements analysis. It is often a matter of the domain expertise of the requirements analyst to detect the inconsistencies in the elicited requirements. If inconsistency is detected during requirements phase, then it can be corrected and the defects which surface later in software development lifecycle can be contained in requirements phase only. If undetected, then the problem permeates the subsequent phases as the programmer might resolve the concern arbitrarily during implementation. The undetected inconsistency is an indication that possibly the analyst has had a presupposition about the inconsistent information, or possibly it was overlooked. Presupposition being a linguistic phenomenon is related to the utterance and the intuitive meaning of the sentence (Levinson, 2000). In context of requirements, presuppositions significantly contribute to both inconsistency and incompleteness problems.

In order to contain defects in the requirements phase and effectively manage the concern of inconsistency, certain degree of formalism is required in the requirements specification. It is essential to draw a formal model or representation of requirements that is a reflection of the expected observable behaviour of the system. Such a model or representation would allow identifying implicit or hidden inconsistency, which is otherwise difficult to identify as implicit inconsistency arises out of interpretation or consequence of the requirements. We can easily identify and take into consideration the explicit inconsistency arising because of multiple views of stakeholders, but no such easy approach is there with implicit inconsistency. We'll see in next section how courteous logic expressions address this requirements representation problem and obligates the analyst to explicitly specify the presuppositions too, if any.

3 COURTEOUS LOGIC BASED SOLUTION

The idea of having requirements representation in a form that shows the expected observable behaviour of the system and the need to formalize requirements specifications led to the use of logical representations of the requirements. We are more focused on non-monotonic reasoning for two reasons. First, to address the problem of inconsistency and explicit specification of presuppositions in the requirements; and secondly, real-world requirements correspond to human way of thinking and common-sense reasoning which is non-monotonic in nature. We found courteous logic based representation (requirements specification hereafter) suitable for our cause. We have already shown the adequacy of courteous logic for requirements specification in (Sharma and Biswas, 2011). Here, we highlight the concern of undetected inconsistency and the associated concern of presupposition.

3.1 Courteous Logic

Courteous Logic (Grosof, 1997) is a form of non-monotonic logic where consequence relation is not monotonic. It is based on prioritization of the rules. Courteous logical representation (CLP) is an expressive subclass of ordinary logical representation (OLP) with which we are familiar and, it has got procedural attachments for prioritized conflict handling. First Order Logic beyond Logic Programming (LP) has not become widely used for two main reasons: it is pure belief language; it cannot represent procedural attachments for querying and actions and, it is logically monotonic; it can not specify prioritized conflict handling which are logically non-monotonic. The Courteous Logic Programming (CLP) extension of LP is equipped with classical negation and prioritized conflict handling. CLP features disciplined form of conflict-handling that guarantees a consistent and unique set of conclusions. The courteous approach hybridizes ideas from non-monotonic reasoning with those of logic programming. CLP provides a method to resolve conflicts that arise in specifying, updating and merging rules. Our CLP representations are based on IBM's CommonRules, available under free trial license from IBM alpha works (Grosof, 2004).

In CLP, each rule has an optional rule label, which is used as a handle for specifying prioritization information. Each label represents a logical term, e.g., a logical 0-ary function constant. The "overrides" predicate is used to specify prioritization. "overrides (lab1, lab2)" means that any rule having label "lab1" is higher priority than any other rule having label "lab2". The scope of what is conflict is specified by pair-wise mutual exclusion statements called "mutex's". E.g., a mutex (or set of mutex's) might specify that there is at most one amount of discount granted to any particular customer. Any literal may be classically negated. There is an implicit mutex between p and classical-negation-of-p, for each p, where p is a ground atom, atom, or predicate.

An example illustrating the power of CLP: Consider following rules for giving discount to customer:

- ❖ If a customer has Loyal Spending History, then give him 5% Discount.
- ❖ If a customer was Slow to pay last year, then grant him No Discount.
- ❖ Slow Payer rule overrides Steady Spender.
- ❖ The amount of discount given to a customer is unique.

These rules are represented in CLP as following set of rulebase:

```
<steadySpender>
if shopper(?Cust) and
   spendingHistory(?Cust, loyal)
then
   giveDiscount(percent5, ?Cust);

<slowPayer>
if slowToPay(?Cust, last1year)
then
   giveDiscount(percent0, ?Cust);

overrides(slowPayer, steadySpender);
```

As discussed above, the two types of customers are labeled as <steadySpender> and <slowPayer>; the predicate 'overrides' is used to override the discount attributed to slowpayer over the discount to be given to steadyspender in case a customer is found to be both steadySpender and slowPayer.

We found that courteous logic representations are closer to natural language representation of business rules in terms of commonly used 'if - then' rules and, can be easily interpreted by both the users and the developers. An experience with CLP shows that it is especially useful for creating rule-based systems by non-technical authors too (Grosof, 1997). Another advantage of CLP is computational scalability: inferencing is tractable (worst-case polynomial time) for a broad expressive case. By contrast, classical logic inferencing is NP-hard for this case.

3.2 Identifying Inconsistency and Presuppositions

In this sub-section, we bring forth the expressive and the reasoning power of courteous logic and, demonstrate how it proves effective in identifying instances of inconsistency and consequent presupposition through three case-studies in varying domains.

<u>Example 1</u> - Representing and Prioritizing Conflicting Views (Academic Grade Processing):

Consider the specification of students' grade approval process where the students' grades are approved by the course-coordinator, the department head and the dean. The *expected behaviour* of the system refers to the fact that at any point in time, approval from department head holds higher priority over course-coordinator; and approval from dean higher priority over department head and in turn, the course coordinator. Often, this observable behaviour is not captured in its essence in requirements specification. The use-case of the academics system that we got to study had a mention of process-flow only as:

The grades of students once entered in the system need to be approved by the course-coordinator, the department head and the dean.

This particular use-case served an excellent example of inconsistency as well as the presence of presupposition as this process-flow nowhere mentions the priority of grade-approval level. In the absence of validation against explicit expected behaviour of the real-time system, the software system can possibly have an inconsistent state of grades subject to the arbitrary implementation done by the programmer. The pragmatic presupposition (Levinson, 2000) associated with this use-case is that when process-flow describes approval by the course-coordinator, the department head and the dean, then it refers to a sequential and prioritized flow with highest priority of the dean, followed by department head and then, course-coordinator. Consequently, programmer needs to take care of these details and should not take any decision arbitrarily.
The courteous logic specification of the requirements as stated in the given use-case translates to four labelled rules, namely new, cdn, hod and dean respectively:

```
<new>
   if assignGrades(?Regno, ?Year, ?Sem,
      ?Group, ?Sub, ?Point)
then  valStatus(new,  ?Regno,  ?Year,
      ?Sem, ?Group, ?Sub);
<cdn>
   if  approvedby(?Regno,  ?Year,  ?Sem,
      ?Group,  ?Sub,  ?Point,  ?Status,
      coordinator)
then  valStatus(coordApproved,  ?Regno,
      ?Year, ?Sem, ?Group, ?Sub);
<hod>
   if  approvedby(?Regno,  ?Year,  ?Sem,
   ?Group,?Sub,  ?Point,  coordApproved,
   hod)
 then   valStatus(hodApproved,   ?Regno,
      ?Year, ?Sem, ?Group, ?Sub);
<dean>
    if   approvedby(?Regno, ?Year, ?Sem,
?Group,?Sub, ?Point, hodApproved, dean)

  then  valStatus(deanApproved, ?Regno,
      ?Year, ?Sem, ?Group, ?Sub);
```

In the expressions above, ?X represents a variable. The rule labelled as <*new*> specifies that when a student with registration number, ?Regno; year of study, ?Year; semester and group as ?Sem and ?Group respectively is assigned grades for points, ?Point in subject, ?Sub, then status of his grades would be *new*. The rule labelled as <*cdn*> specifies that grade status changes to *coordapproved* on approval by coordinator. Similarly, the rules labelled <*hod*> and, <*dean*> indicate the grade status on approvals by the department head and the dean respectively.

We observed the expected behaviour of the system by subjecting the courteous logic representation of this use-case to reasoning engine of courteous logic. We first collected observation in the absence of any prioritization rules as that information was not explicitly mentioned in the given use-cases. This use-case presented us with the case of implicit inconsistency present in the consequences of the above rules. We took three sample given facts for a student with registration number 2008CSY2658:

```
   assignGrades(2008CSY2658,       2009,
even, 4, ai, c);
    approvedby(2008CSY2658,  2009,  even,
    4, ai, c, new, coordinator);
   approvedby(2008CSY2658,  2009,  even,
4, ai, c, new, hod);
```

Corresponding to these facts and the given labeled rules, the consequences inferred for grade status of this student was found to be taking three distinct values at one point in time:

```
   valStatus(hodApproved,   2008CSY2658,
2009, even, 4, ai);

   valStatus(new,    2008CSY2658,   2009,
even, 4, ai);

   valStatus(coordApproved,
2008CSY2658, 2009, even, 4, ai);
```

Since this is not practical for any status term to have multiple values assigned, it represents an inconsistent state of the world. For all practical purposes, we can safely say that above specification is an *inconsistent reflection of the real-world system.* This is an implicit inconsistency, having occurred owing to the consequences of the requirements expressed as labelled rules. Since our specification is an executable model of the real-world, we could validate the specification against expected behaviour of the system and, reach the conclusion of inconsistency at an early stage of software development. Next, this observation pointed to the presence of some knowledge which is not yet put into words, i.e. *presupposition*. Further investigating and refining the requirements based on this observation and enriched knowledge, we added following rules:

```
   overrides(cdn, new);
   overrides(hod, new);
   overrides(dean, new);
   overrides(hod, cdn);
   overrides(dean, cdn);
   overrides(dean, hod);

   MUTEX
      valStatus(?Status1,       ?Regno,
      ?Year, ?Sem, ?Group, ?Sub)
      AND
      valStatus(?Status2,       ?Regno,
      ?Year, ?Sem, ?Group, ?Sub)
   GIVEN
     notEquals( ?Status1, ?Status2 );
```

The overrides clause establishes the prioritizing relationship between the grade approval rules where the first argument holds higher priority above the second argument. The MUTEX specifies the scope of conflict, which is the grade status in our case. The overrides clause takes care of the possible conflict in the student's grade status. Validating the updated specification against the observable expected behaviour of the grade approval processing, we found our specification consistent as the consequence obtained was the expected one:

```
   valStatus(hodApproved,  2008CSY2658,
2009, even, 4, ai);
```

The above example illustrates how conflicting information can be expressed with well-formed semantics of courteous logic. We present one more example below that illustrates expressing some default operation in a domain as well as exceptions to that processing.

***Example 2* – Representing Default and Exceptional Scenario Processing (Saving and Current Account Processing):**

Consider the account processing part of a bank customer where he can have more than one account. Let's consider that a bank customer can have a current account and a saving account. The customer can choose one of these accounts as default account for any transaction that he wants to carry out. The usual choice is current account but to keep the use-case generic, let us assume that customer has marked one of the accounts as default. The customer is free to select the other account for some of his transactions. In that case, the selected account processing should override the default processing. The natural language expression for such default operation and associated exception can be easily understood by the involved stakeholders as well as developers. But what is often overlooked by developers is the implicit interpretation here – the account chosen for default processing should remain unaffected in case selection is made for the non-default account and often, this is uncovered till testing phase. Such overlooked implicit interpretation results in implicit internal inconsistency. Such a defect can be easily detected during RE phase if we have an executable model or representation of requirements that can sufficiently express the domain knowledge.

The courteous logic specification of the requirements as stated in the given account processing for deposit and withdrawal transactions by the customer translates to following rules:

```
   <def>
    if deposit(?Txn, ?Client, ?Amount)
    and      holds(?Client,      ?Acct)
    and default(?Acct)
   then    addAmount(?Client,     ?Acct,
?Amount);
   <sel>
    if deposit(?Txn, ?Client, ?Amount)
    and   holds(?Client,   ?Acct)   and
    option(?Client, ?Txn, sel, ?Acct)
   then    addAmount(?Client,     ?Acct,
?Amount);
   <def>
    if withdraw(?Txn, ?Client, ?Amount)
    and   holds(?Client,   ?Acct)   and
    default(?Acct)
   then    subAmount(?Client,     ?Acct,
?Amount);
    <sel>
    if withdraw(?Txn, ?Client, ?Amount)
    and   holds(?Client,   ?Acct)   and
    option(?Client, ?Txn, sel, ?Acct)
    then    subAmount(?Client,     ?Acct,
    ?Amount);
```

The rule with label <*def*> indicates transaction processing from default account whereas the rule with label <*sel*> indicates processing from the selected account. For deposit type of transaction, the default rule (the first rule in the above expressions) indicates that if a client, ?Client is holding an account, ?Acct marked as default and he initiates a deposit transaction with a certain amount, ?Amount then that amount will get credited or added to his default account. Similar is the case with withdrawal transaction. The client can choose another account provided he is holding that account and chooses it for some transaction as expressed through rule labeled as <*sel*>. In this case, any deposit or withdrawal transaction would affect the selected account only. To verify that any such transaction will not affect the default account, we first tested the rules without any override clause using some sample data as:

```
    holds(abc, acctabc10);
    holds(abc, acctabc11);
    default(acctabc10);
    deposit(t1, abc, 1000);
    deposit(t2, abc, 5000);
    withdraw(t3, abc, 2000);
    withdraw(t4, abc, 4000);
    option(abc, t2, sel, acctabc11);
    option(abc, t3, sel, acctabc11);
```

Corresponding to these facts and the given labeled rules, the consequences inferred for each of the transactions t1, t2, t3 and t4 were found to be:

```
    addAmount(abc, acctabc10, 1000);
    addAmount(abc, acctabc10, 5000);
    addAmount(abc, acctabc11, 5000);
    subAmount(abc, acctabc10, 2000);
    subAmount(abc, acctabc11, 2000);
    subAmount(abc, acctabc10, 4000);
```

The results obtained indicate that deposit to and withdrawal from default accounts are adequately expressed as the behavior of dummy transactions, t1 and t4 is same as the expected behavior. But on account selecting, both the selected and the default accounts are getting affected as transactions t2 and t3 represent the outcome of the requirement expression. Adding the clause for prioritizing the selected account and making the default account and the selected account mutually exclusive so that only one of these accounts is impacted by some operation, we got the desired output – one that matches the expected behavior in real-time scenario:

```
   overrides(sel, def);
   MUTEX
    addAmount(?Client, ?Acct1, ?Amount)
   AND
    addAmount(?Client, ?Acct2, ?Amount)
   GIVEN
      notEquals( ?Acct1, ?Acct2 );

   MUTEX
    subAmount(?Client, ?Acct1, ?Amount)
   AND
    subAmount(?Client, ?Acct2, ?Amount)
   GIVEN
        notEquals( ?Acct1, ?Acct2 );
```

As elaborated in detail in example 1, the mutex clause in this case establishes the scope of conflict over the two accounts and, the override clause assigns priority to the selected account. This example highlights two things:

a) The labels used as handle to some rule are not mere tags that need to be different from each-other. These can be repeated and reused in same specification provided their intent is same wherever used.

b) The expressions without 'override' and 'mutex' clause were consistent with natural language specification of the requirements but, were inconsistent with expected behavior. The implicit presupposition that only the selected account should be affected was uncovered at requirements level owing to the executable nature of our specification.

Let's consider one more example to show that multiple views of stakeholders can also be conveniently expressed using courteous logic based requirements specifications.

<u>Example 3</u> – Representing and Prioritizing Views of Multiple Stakeholders (Corporate Event Processing):

Consider a corporate action event announced on a security. If a client is holding the security on which event is announced, then that client is eligible to get the announced benefits of the event. These benefits can either be in the form of cash or stock or both. The types of benefits disbursed to the clients vary from one event type to another; it also depends on various other factors like base country of the security on which event is announced, the country of the customer; client opting for an option etc. Then, there can be multiple stakeholders having differing views like one particular stock market has rules that do not allow client to opt any option announced on event; whereas, clients from some other market can opt for event's announced operations, so on and so forth. We took a small subset of this large set of rules and gradually scaled the rules as well as the data to find that results are consistent with the actual observable expectations. This particular example served towards claiming scalability of courteous logic based requirements specifications. Our expressions could not only be easily validated against the real-world expected behavior but also these were small and compact making them easy to comprehend and verify against multiple real-world scenarios as shown below:

```
    <cash>
    if event(?EventId, ?Type, ?Security)
    and holds(?Client, ?Security) and
    opts(?Client, cash)
   then distribute(?Client, ?EventId,
cash);
   <stock>
```

```
    if event(?EventId, ?Type, ?Security)
    and holds(?Client, ?Security) and
    opts(?Client, stock)
   then distribute(?Client, ?EventId,
stock);
```

```
   <both>
    if event(?EventId, ?Type, ?Security)
    and holds(?Client, ?Security) and
    opts(?Client, both)
   then distribute(?Client, ?EventId,
both);

   <divMtk1>
    if event(?EventId, dividend,
    ?Security) and holds(?Client,
    ?Security) and baseCntry(?Security,
    Mkt1)
   then distribute(?Client, ?EventId,
stock);
```

```
   <divMkt2>
    if event(?EventId, dividend,
    ?Security) and holds(?Client,
    ?Security) and clientCntry(?Client,
    Mkt2)
   then distribute(?Client, ?EventId,
nothing);
```

```
    <divMkt1Mkt5>
    if event(?EventId, dividend,
    ?Security) and holds(?Client,
    ?Security) and baseCntry(?Security,
    Mkt1) and clientCntry(?Client, Mkt5)
   then distribute(?Client, ?EventId,
cash);
```

The rule with label as <*cash*> indicates that if an event, ?Event of some type, ?Type is announced on a stock, ?Security and a client, ?Client is holding that stock and he opt for '*cash*' option then he will receive the benefit of event in the form of cash as per the announced rules of the event. Similarly, use-cases with stock and both types of disbursements are represented through rules labelled as '*stock'* and '*both'* respectively. These are generic rules. Next, we have considered a hypothetical scenario where in stakeholders from stock market, *Mkt1* are of the view that if 'dividend' type of event is announced on the stock belonging to their nation, then all customers shall get event's benefits as stock only. This is represented in the rule labeled as <*divMkt1*>. The rule with label <*divMk2*> indicates that dividend event announced will not entail any benefits to clients from stock market *Mkt2*. The last rule is an exception to rule <*divMkt1*> - it says that if client hails from the stock market, *Mkt5* then he is eligible for benefit in the form of cash rather than stock.

The above-mentioned rules were then verified against facts from real-world as below:

```
event(11, dividend, samsung);
event(22, dividend, dell);
baseCntry(dell, US);

holds(abc, samsung);
holds(abc, dell);
holds(xyz, dell);
holds(pqr, dell);
clientCntry(xyz,Mkt2);
clientCntry(pqr, Mkt5);
opts(abc, both);
```

In the absence of any kind of prioritization amongst multiple views, we got the validation results as:

```
distribute(pqr, 22, cash);
distribute(pqr, 22, stock);
distribute(xyz, 22, nothing);
distribute(xyz, 22, stock);
distribute(abc, 11, both);
distribute(abc, 22, both);
distribute(abc, 22, stock);
```

These results are not in line with what actual happens in the stock market as one conclusion indicates no benefit to xyz for event 22; whereas next conclusion points out stock benefit to the same client on the same event. When the multiple views from stakeholders of different stock market were assigned priorities (that can be easily modified or updated later on too), the results obtained were as per the expected benefits disbursed to the client in stock market abiding terms and conditions:

```
overrides(divMkt1Mkt5,divMkt2)
overrides(divMkt1Mkt5,divMkt1)
overrides(divMkt1Mkt5,cash)
overrides(divMkt1Mkt5,stock)
overrides(divMkt1Mkt5,both)
```

and similar rules for rest of the markets including the generic ones:

```
overrides(both,stock);
overrides(both,cash);
overrides(stock, cash);
MUTEX
 distribute(?Client,?EventId,Value1)
AND
 distribute(?Client,?EventId,?Value2)
GIVEN
     notEquals( ?Value1, ?Value2 );
```

Validating the facts gathered earlier against the set of labeled rules and the prioritized information, consistent and expected results were obtained as:

```
distribute(abc, 22, stock);
distribute(pqr, 22, cash);
distribute(abc, 11, both);
distribute(xyz, 22, nothing);
```

3.3 Observations

Courteous logic based representations can sufficiently express software requirements with reasoning and inferring mechanism as shown in (Sharma and Biswas, 2011). We have explored the inconsistency and presupposition concern in detail in this paper. The advantage of the courteous logic based requirements specification lies in following observations:

1. It supports expressing conflicting information in a way that is subjected to prioritized conflict-handling. Any modification to the priority rules is a matter of changing the priorities.
2. Adding any information or removing any information during requirements evolution ensures consistency of the requirements specification.
3. Having inference mechanism based FOL, it allows validating the specifications against the expected observable behavior of the system.
4. It provides assistance in identifying presuppositions (tacit knowledge) in the specified requirements.
5. Earlier detection of defects as well as any disconnect between the client's intent and the requirements analyst's interpretation.

We also observe that courteous logic based requirements specifications have the potential to improve requirements elicitation, management and evolution. Point 5 leads to further noticing that earlier detection of problems can considerably reduce cost and time effort of the project and help in keeping project schedule on time. Further, courteous logic based requirements specifications can be helpful in identifying and preparing test-cases. The test-cases can be easily mapped to the rules present in the courteous logic based requirements specifications. The validation data can serve as the starting base to test data.

4 RELATED WORK

The use of logic for requirements representation and resolving inconsistency issue in software requirements has been acknowledged earlier too and has found its place in several authors' work. RML (Greenspan, Borgida and Mylopoulos, 1986) is one of the earliest logic-based formal requirements modeling language. Revamping efforts on RML gave way to two more related languages – Conceptual Modeling Language (CML) (Stanley, 1986) and Telos (Mylopoulos et al., 1990). HCLIE language (Tsai and Weigert, 1991) is a predicate logic-based requirement representation language. It makes use of Horn clause logic, augmented with multiple inheritances and exceptions. HCLIE handles non-monotonic reasoning by not reporting any consequence in case any conflict arises. Description Logic has also been used for representing requirements in (Zhang and Zhang, 2007). Description logic is an expressive fragment of predicate logic and is good for capturing ontology whereas, Horn clause logic is useful for capturing rules. Ordinary Logic Programming (OLP) lacks in conflict-handling or non-monotonic reasoning. Default Logic form of non-monotonic reasoning has also been used to reason about inconsistencies in requirements in (Gervasi and Zowghi, 2005). The computational complexity of default logic is high and the expressions are not too easy and convenient to comprehend.

Business rules do present themselves with many instances of conflicts and presuppositions around them. Handling business rules along with constraints and related events as well as the semantics of rules are of paramount importance in business processes. To capture business rules successfully, Horn clause logic needs to be augmented with non-monotonic reasoning as has been discussed in (Borgida, Greenspan and Mylopoulos, 1985) and, here we have presented courteous logic based representations towards the cause. Syntactical presuppositions have been addressed in (Ma, Nuseibeh and et.al., 2009). We have presented logical expressions as a solution towards addressing semantic and pragmatic presuppositions. The underlying essence and rational of requirements and the corresponding contextual meaning can be understood via some executable model of requirements. We have presented such a model in the form of courteous logic based requirements representation.

5 THREATS TO VALIDATION

The logical expressions do offer a mechanism to reason with the requirements and resolve relevant issues. Nevertheless, these formal logical expressions might pose usability problems to the stakeholders as well as those developers who are not fluent with the formal logic.

Even though the courteous logic representations are more English-like with less usage of arrows and symbols, still accepting a new approach is not very encouraging until its benefits are realized. Secondly, some amount of training would have to be imparted to the involved parties in requirements phase in order to make them understand the syntax and the querying mechanism. Third, translating the requirements corpus to the courteous logic expression would be time-consuming and would depend on the individual's expertise and skills. Automated conversion from natural language to these representations would certainly be an advantage but doing that itself poses natural language parsing challenge.

We still hope that benefits drawn from using non-monotonic formal representations of requirements will outweigh the threats. Requirements of varying domains are of varied nature and not one kind of model is able to sufficiently express all the aspects of the system. A domain which is rule-intensive and has multiple conflicting views and requirements would certainly be benefitted by courteous logic based specifications.

6 CONCLUSIONS

In this paper, we have successfully addressed the problem of identifying and analyzing the logical inconsistency in the software requirements using courteous logic based requirements specifications in our work. We have shown that these specifications, being an executable model of the system's requirements, not just identify implicit inconsistencies but also help in identifying and specifying presuppositions explicitly. The results from the case-studies are encouraging. Tractable inferencing and scalability of the representations are some of the motivating factors towards using courteous logic based specifications. We have also demonstrated that fixing the inconsistency by rule-prioritization does not entail a major change in the existing requirements specification. This aspect makes these expressions a suitable choice from software maintenance point of view. Though the requirements analysts might not find the idea of using courteous logic comfortable, but since these representations are relatively simpler; easy to comprehend and natural language like, we hope that with small amount of training these could be well-taken by the practitioners as well.

We further aim to refine the current proposed requirements representations and incorporate the second interpretation of inconsistency – traceability; and then develop a semantic framework for automated analysis of requirements. We see our framework as a foundation towards integrated framework for semantic software engineering.

REFERENCES

IEEE Computer Society, 1998. IEEE Recommended Practice for Software Requirements Specification, IEEE Std 830 – 1998(R2009).

Zowghi, D., 2003. On the Interplay Between Consistency, Completeness, and Correctness in Requirements Evolution. In *Information and Technology*, Vol 45, Issue 14, 2003, pp 993-1009.

Ma, L., Nuseibeh, B., Piwek, P., Roeck, A.D. and Willis, A., 2009. On Presuppositions in Requirements. In *Proc International Workshop on Managing Requirements Knowledge*, pp.68–73

Tsai, J. J. P., Weigert, T. and Jang, H.,1992. A Hybrid Knowledge Representation as a Basis of Requirements Specifciation and Specification Analysis. *In IEEE Transaction on Software Eng*g, vol. 18, No 12, 1992, pp. 1076–1100.

Gervasi, V. and Zowghi, D., 2005. Reasoning about Inconsistencies in Natural Language Requirements. In *ACM Transactions on Software Engg and Methodology*, Vol 14, No 3, 2005, pp. 277-330.

Levinson, S. C., 2000. *Pragmatics*, Cambridge University Press.

Sharma, R. and Biswas, K.K. 2011. Using Courteous Logic based representatiosn for Requirements Specifications. In *International Workshop on Managing Requirements Knowledge*.

Grosof, B.N., 1997. Courteous Logic Programs: prioritized conflict handling for rules. *IBM Research Report RC20836, IBM Research Division, T.J. Watson Research Centre.*

Grosof, B.N., 2004. Representing E-Commerce Rules via situated courteous logic programs in RuleML. *Electronic Commerce Research and Applications*, Vol 3, Issue 1, Spring 2004, pp 2-20.

Greenspan, S., Borgida, A. and Mylopoulos, J.,1986. A Requirements Modleing Language and its logic, *Information Systems*, vol 11, no 1, 1986, pp 9-23.

Stanley, M., 1986. CML: A Knowledge Representation Language with Applications to Requirements Modeling, *M.Sc. Thesis, Dept Comp. Sc.*, University of Troronto.

Mylopoulos, J., Borgida, A. and Koubarakis, M.,1990. Telos: Representing Knowledge about Information Systems, *ACM Transactions on Information Systems*, 1990.

Tsai, Jeffrey J.-P. and Weigert, T.,1991. HCLIE: a logic-based requirement language for new software engineering paradigms, *Software Engineering*, vol 6, issue 4, July 1991, pp 137-151.

Zhang, Y. and Zhang W., 2007. Description Logic Representation for Requirement Specification, *Proc International Confernce on Computational Science (ICCS 2007)*, Part II, Springer-Verlag, pp 1147 – 1154.

Borgida, A., Greenspan, S. and Mylopoulos, J., 1985. Knowledge Representation as the basis for Requirements Specifications, *Computer*, vol 18, no 4, Apr. 1985, pp 82-91..