Using Courteous Logic based representations for Requirements Specification

Richa Sharma
School of IT
IIT Delhi
India
sricha@gmail.com

K.K. Biswas
Dept. of Computer Science
IIT Delhi
India
kkb@cse.iitd.ac.in

Abstract—**The growing complexity and size of software systems emphasize the need for capturing the requirements in a way that is amenable to requirements validation and facilitates requirements management and evolution. Knowledge Representation techniques have widely been used for representing the requirements, each with varying degree of success. In this paper we present courteous logic based representations for specifying the requirements. We explain how courteous logic can be used to represent the requirements, resolve inconsistencies, incompleteness, ambiguities and presuppositions in elicited requirements and present solution to requirements management and evolution problem.**

Keywords- requirements specification; courteous logic; knowledge representation; requirements validation

I. INTRODUCTION

Requirements Specification serves an important role during the initial phase of development of an information system. Being a representation of users' and stakeholders' goals and expectations, requirements specification acts as a communication link between the analysts and the developers, and supports requirements analysis, validation and evolution. It forms the basis for subsequent phases of software development lifecycle including design, implementation, testing and maintenance phase. Therefore, it is important that requirements specifications should satisfy certain desirable features to meet the expected objectives.

The desirable features for requirements specification language have been proposed by numerous authors (like [1], [2], [3], [4]) independently at different times. Most of the proposed features are common in nature and there are overlaps in these different descriptions.

The Section II in this paper discusses the desirable features of requirements specification language as proposed earlier and a consolidated set of those. In section III, we present an overview of courteous logic and its application for requirements representation. In Section IV, we explain how this representation conforms to the desirable features taking a case-study. Section V discusses the related work and finally, section V1 presents the conclusion.

II. DESIRABLE FEATURES OF REQUIREMENTS SPECIFICATION LANGUAGES

While representing requirements for an information system using requirements representation language, the analyst is interested in describing the behavior of the system in terms of the business users' and stakeholders' expectations from the proposed information system. But, the representations should be well understood by the system designers and developers too. The representations should be analyzable too and it's important to decide the level of details included. Analyst is, therefore, posed with multiple challenges and an important question: what should be the desirable features of the requirements representation (requirements specification hereafter). We give an overview of some of the recommendations from various authors and then draw a consolidated set of features.

In [1], Zualkernan and Tsai have carefully distinguished between problem specification and system specification. The problem specification is a statement of a problem that a client has. A system specification, on the other hand, is a description of a system that can solve the problem by achieving goals, given the constraints and the exploiting the constraints on types if available knowledge. Both of these specifications are interdependent and an analyst is supposed to draw upon problem specification while representing requirements; he might need to include some parts of system specification though at times to bring more clarity to the specifications. Zualkernan and Tsai have suggested some of these important criteria for a language to qualify for problem specification:

1. **Epistemological Adequacy** – Also called as expressive power in context of knowledge representation and, refers to the adequacy of a formal system to represent concepts in real-world.
2. **Synthesis Adequacy** – It refers to the criteria of maintainability from representation perspective and, availability of procedures for constructing specification from methodology point of view.
3. **Analysis Adequacy** – It refers to the adequacy of representation to yield itself to validation and verification.
4. **Contractual Adequacy** – This is the adequacy of the representation to support interaction with domain experts or clients.
5. **Blue-print Adequacy** – This is the adequacy of problem specification to serve as blue-print for next stages of development.

Tsai and Weigert [2] have proposed that if the requirements can be operationally interpreted, then the user would be exposed to a 'working model' of the system at an

978-1-4577-0938-8/11/$26.00 ©2011 IEEE

early stage of development. Secondly, if an inference mechanism is available for the requirement specification, then it can be used to validate the requirements. They propose following demands on a language to express requirements:

1. Free from implementation concerns.
2. Ability to check the validity of the requirements.
3. Ability to express both the objects and the relations of the domain, as well as the constraints on them and rules about them in a straightforward manner.
4. Should be sound and complete
5. Should allow for operational interpretation of the system behavior.

Tse and Pong in [3] have recommended following points of consideration for requirements specification:

1. **Abstraction of the real-world** – The requirements language must allow abstracting out the important issues from non-essentials. This will help in improving the conceptual clarity of the problem.
2. **Manipulation of the requirements** – Requirements specification must be structured in such a way that parts of it can be modified as requirements gradually evolve. Requirements should be expressible in a precise notation with a unique interpretation to avoid ambiguity.
3. **Construction of real-world system** – Requirements specification should be independent of the design and implementation issues. The elements in specification should be traceable to the final design and implementation.

In [4], the recommendation is to deal with over-abstraction and ensuring consistency of the model. It is further proposed that AI research on non monotonic logics may provide insight on how to make final specification logically consistent. Our focus in this paper is to exploit non-monotonic logic for constructing logically consistent requirements specifications.

There are many a instances where desirable features of requirements specification have been discussed and, most of these refer to more or less similar features as discussed above. A careful look at above discussed features too shows overlaps. We aim at achieving consolidated set of these features in our proposed courteous logic based requirements specification. We have consolidated the desirable features as:

1. The requirements specification should be an abstract representation or model of the real-world.
2. The requirements specification should not be affected by the design and development of the information system.
3. It can be subjected to validation and verification in terms of the observable behavior of the real-world, i.e. an inference mechanism should exist.
4. It should be sound and complete. Any modification to existing requirements should not result in an inconsistent set of requirements.
5. The requirements specification should be maintainable and traceable to design and development artifacts.
6. It should be able to act as a bridge between the user or client and the development team, i.e. it should be well understood by both parties.

Of the several approaches to formalize the requirements languages, logical representations have been widely acknowledge with the increasing complexity and expectations of software systems. Our focus is on non-monotonic logic as real-world requirements correspond to human way of thinking and common-sense (non-monotonic) reasoning. These aspects require that any logical specification of requirements should be able to handle non-monotonic reasoning.

III. COURTEOUS LOGIC BASED REPRESENTATION OF REQUIREMENTS

A. Non-monotonic L ogic

Non monotonic logic refers to formal logic where consequence relation is not monotonic. An important goal of knowledge representation and reasoning is to reach only the true conclusions and, we also require that our conclusions are justified. These two constraints are not same. Justification preserving is not always monotonic and human reasoning too is not monotonic; and, therefore the need for non monotonic logic. There are several forms of non-monotonic logics. Default logic, proposed by Raymond Reiter [5] is one of the oldest and most studied non-monotonic logic. Defeasible Logic, proposed by Donald Nute [6], is based on strict rules, defeasible rules and defeaters. Conclusions are tentative in the sense that a conclusion can be withdrawn when there is a new piece of information. Courteous Logic [7] is based on prioritization of the rules. This is in contrast to defeasible logic where rules are marked as strict and defesible rules. We found that courteous logic representation is more suitable for our cause for two main reasons: first, it supports non-monotonic reasoning; and, second, the representation can be well understood by the involved parties during the requirements phase of software development. We'll discuss features of courteous logic in next subsection.

B. Courteous L ogic

Courteous logical representation is an expressive subclass of ordinary logical representation 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 [7]. 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 [8].

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. 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 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 [7]. We present a case-study in the following section with the idea of showing that courteous logic based requirements specification satisfies the desirable features of requirements specification language. We were able to uncover ambiguities, resolve inconsistency and presuppositions in the studied requirements set using courteous logic based requirements specifications.

IV. CASE-STUDY

We studied the registration, grade-processing and graduation part of an educational institute to carry out our study. The reason behind choosing these two processing parts was that these are rule-intensive and we took the challenge of writing the given use-cases using courteous logic representations and then, ensure that consistency between rules is preserved. We were presented processing information in the form of use-cases and here we'll take the running example of grade-processing part.

Our aim in conducting this experiment was to determine if courteous logic based specifications meet the desirable features as discussed in section-II.

The experiment started with the identification of entities, relevant attributes and relations in the given use-cases. We identified a total of 8 entities along with their attributes in this sub-system. We translated the processing part of the use-cases under study to courteous logic representation. Much to our satisfaction, it didn't turn out to be an arduous task. The information obtained was already structured in the form of use-cases and secondly, the translation part didn't require much experience with CommonRules. We'll now take each point from desirable features of requirements specification language covered in section-II:

1. Abstract representation of real-world

Within the periphery of given use-cases, identifying the entities; their attributes and relations was not ambiguous. We could represent the behavior of the system by capturing the governing rules and the constraints imposed at a high level without delving into any sort of implementation details. This representation itself worked as a 'working model' for the subsystem under study. This point would be clearer with the scenario presented in next point.

2. Not affected by design and development details

The requirements are modified only when something new is encountered at a later stage. In our case, we didn't have to tweak our requirements specifications later as the development details started pouring in. An interesting and relevant scenario was with grade conversion use-case:

The Grade Conversion use-case was defined with preconditions stating:

Grade rules must be defined in the system and grades should be submitted and approved.

The processing part stated:

User initiates 'Grade Conversion' process. User selects academic year, semester and course of which he wants to change the grade. User then selects the student names and can optionally give remarks while converting the grade. The updated information is saved.

The post-condition part stated:

Grade status gets modified and entire workflow of grade approval is initiated.

We represented this use-case as:

<gradeMod>
if convertGrades (?Regno, ?Year, ?Sem, ?Group, ?Sub, ?Status, ?OldPoint, ?NewPoint)
then valStatus (new, ?Regno, ?Year, ?Sem, ?Group, ?Sub);

The representation indicates that any grade conversion request for a student with registration Id, *Regno;* year as *Year*; semester and group as *Sem* and *Group* respectively; with subject and current grade status as *Sub* and *Status* respectively would change the status of grades to '*new*', which calls for grade approval process as is the case with new grade assignment:

<new>
if assignGrades(?Regno, ?Year, ?Sem, ?Group, ?Sub, ?Point)
then valStatus(new, ?Regno, ?Year, ?Sem, ?Group, ?Sub);

The design team preferred to have grade approval part as a component reusable at the time of regular/new (without conversion) grade approval and at the time of grade conversion. The development team, on the other hand, couldn't use that component in its entirety owing to technical reasons and had to go by the choice of separate components. We have two observations here to make:

a. The representation remained unaffected by the choices design team and development made. It just represented the abstracted underlying idea that grade change would entail invoking the grade approval processing right from the very beginning.
b. The abstract idea had a suitable representation in the use-case too but courteous logic representation was quite compact as compared to the use-case one. Secondly, we could verify the observable behavior of grade change as courteous logic specifications represented working model of the system.

3. Validation in terms of observable behavior of the system

We have already made a reference to this observation in the above point that courteous logic based specifications represent working model of the system. We would like to explore this observation in terms for requirements analysis for two main issues that analysts encounter namely: ***resolving ambiguity and presuppositions*****;** and ***finding incompleteness*****.**

Direct Grade Change use-case was defined with Administrator as actor and preconditions stating:

Grade rules must be defined in the system and grades should be submitted and approved.

The processing part stated:

User initiates 'Direct Grade Change'. User selects academic year, semester and student whose grades need to be changed. A list is displayed showing the student's grades in the respective courses he enrolled. User submits the information to save.

The post-condition part stated:

Grade submitted are finalized.

A quick look at the use-case description apparently doesn't pose any problem. But, when this use-case was studied with other use-cases where three different levels of approval are defined, we were immediately posed with these questions:

a. When can admin intervene for direct grade change?
b. What is meant by grade finalization?
c. Do finalization and approval refer to same grade status? What happens if there is no direct grade change request? How would finalization of grades happen?
d. Can this use-case be executed again after finalization?

It's quite possible that different people are working on different use-case and may not uncover incompleteness in their use-case in context of the environment. This use-case had ambiguity in terms of approval and finalization status and, this was overlooked in the textual use-case. When we approached subject matter expert (SME hereafter), then we found that there was a presupposition in the grade-processing use-cases. It was assumed that approval by dean refers to finalization of grades and that the above use-case would be executed in exceptional scenario only when grades are finalized (that called for a change in precondition too). In such a scenario the system administrator will do the needful as mentioned in this use-case.

We could successfully find above-mentioned ambiguity, presupposition and incompleteness at the time of requirements specification only.

4. Ensuring consistency

This is the main focus point that motivated us to make use of courteous logic based representation of requirements. It's important while representing and analyzing the requirements that any modification to current requirements should not result in any sort of inconsistency.

We would like to bring out this point following the same use-case as discussed in point-3. Our representation had already captured grade approval part and the facts whose approval would be given priority as:

When we filled for the gaps in the use-case as discussed in point-3, we could easily modify the requirements without any inconsistency creeping in as; courteous logic representation has already provided means of prioritized conflict handling. We could also validate the observable behavior of the system by executing different scenarios. Existing requirement specifications were:

<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);
<gradeMod>
if convertGrades(?Regno, ?Year, ?Sem, ?Group, ?Sub, ?Status, ?OldPoint, ?NewPoint)
then valStatus(new, ?Regno, ?Year, ?Sem, ?Group, ?Sub);
overrides(cdn, new);
overrides(hod, new);
overrides(dean, new);
overrides(hod, cdn);
overrides(dean, cdn);
overrides(dean, hod);
overrides(gradeMod, cdn);
overrides(gradeMod, hod);
overrides(gradeMod, dean);

Adding updated information for 'Direct Grade Change' use-case entailed specifying:

<adminMod>
if convertGrades(?Regno, ?Year, ?Sem, ?Group, ?Sub, deanApproved, ?OldPoint, ?NewPoint)
then valStatus(deanApproved, ?Regno, ?Year, ?Sem, ?Group, ?Sub);

5. Maintainability and Traceability

As we've made our observation while discussing point 2 that courteous logic based requirements representation is quite compact and preserves consistency (refer point 4), it can be said that this representation is maintainable in nature. We're further planning to refine the proposed representations to take care of traceability as well.

6. Well understood by the involved parties

While carrying out our case-study, we worked in close collaboration with the analysts and the developers of the system. We had to check with SMEs on account of any gap or ambiguity and then, we took our representation to developers. We found that none of the parties had much difficulty in making sense of these representations. Developers found these quite helpful as it was easier for them to relate observable behavior of the system to the expected behavior. This observation led us to believe that our approach should turn out to be practical, though we have to further test the representations for practicability.

V. RELATED WORK

The use of use of logic for requirements representation has been acknowledged earlier too and has found its place in several authors' work. RML [9] is one of the earliest logic-based formal requirements modeling language. HCLIE language [2] is a predicate logic-based requirement representation language. It makes use of Horn clause logic, augmented with multiple inheritances and exceptions. Description Logic has also been used for representing requirements in [10]. 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. Business rules do present themselves with many instances of conflict handling along with constraints and related events. To capture business rules successfully, Horn clause logic needs to be augmented with non-monotonic (/commonsense) reasoning as has been discussed in [4] and, here we have presented courteous logic based representations towards the cause.

VI. DISCUSSION AND CONCLUSION

Our preliminary work has shown that courteous logic based requirements representation meets the consolidated set of desirable features for a requirements specification language. We presented scenarios in our case-study to show how this representation can prove helpful in ensuring consistency and resolving conflicts, presuppositions and incompleteness in the elicited requirements. We now further aim to refine the representations to include traceability and, develop framework for formal analysis of requirements as part of our future work.

REFERENCES

[1] I.A. Zualkernan and W.T. Tsai, "Are Knowledge Representations the answer to Requirements Analysis," Proc IEEE Conference on Computer Languages (ICCL 1988), IEEE Press, Oct. 1988, pp 437-443, doi:10.1109/ICCL.1988.13094.

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

[3] T.H. Tse and L. Pong, "An examination of requirements specification languages," The Computer Journal, vol 34, issue 2, Apr. 1991, doi:10.1093/com/jnl/34.2.143

[4] A. Borgida, S. Greenspan, and J. Mylopoulos, "Knowledge Representation as the basis for Requirements Specifications," Computer, vol 18, no 4, Apr. 1985, pp 82-91, doi:10.1109/MC.1985.1662870

[5] R. Reiter, "A logic for default reasoning," Artificial Intlligence, vol 13, 1980, pp 81-132.

[6] D. Nute, "Defeasible Logic," Proc International Conference on Applications of Prolog (INAP 2001), IF Computer Japan, 2001, pp 87-114.

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

[8] B.N. Grosof, "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, doi:10.1016/j.elerap.2003.09.005.

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

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