On Pronouncements of Requirements Oracles

Ivan J. Jureta
FNRS & Louvain School of Management
University of Namur
ivan.jureta@fundp.ac.be

Alex Borgida
Dept. Computer Science
Rutgers University
borgida@cs.rutgers.edu

***Abstract*—A requirements database contains the information elicited or otherwise collected, used, and produced during requirements engineering. From a requirements database, the requirements oracle selects information relevant for the resolution of the requirements problem, such as which goals are satisfied, which domain assumptions are violated or maintained, which tasks cannot be executed together. Implemented oracles should enable the resolution of the requirements problem to become an interactive process, in which the oracle provides feedback about the effects of the changes in the requirements database on the candidate solutions to the requirements problem. We give a definition of the pronouncements of requirements oracles by formally defining their desirable properties.**

***Keywords*-decision support, knowledge base, requirements oracle**

I. Introduction

Suppose that we have a database of requirements (denote it Δ) for an ecommerce web application and that it includes the following pieces of information:

$\mathbf{t}(p_1)$: Display a link to the shopping cart on every screen of the web application.

$\mathbf{k}(p_2)$: If the link to the shopping cart is displayed on every screen, then the user can view the summary of the shopping cart at any time.

We use the label $\mathbf{t}()$ on the proposition p_1 to say that p_1 refers to a task, i.e., to display a link to the shopping cart on every screen of the web application is a task that needs to be executed. The label $\mathbf{k}()$ says that p_2 is an assumption that we make when engineering the requirements of such a system.

While Δ does record these requirements, it alone tells us nothing useful regarding how close we are to finding a solution to the requirements problem, i.e., finding tasks and assumptions that need to be executed and maintained by the system-to-be in order for it to satisfy its goals and quality constraints. This makes it difficult to know what we should add to, or remove from Δ.

If the database Δ is combined with a program that can compute logical consequences from the database, and since we can write $\mathbf{k}(p_2)$ as $\mathbf{t}(p_1) \rightarrow \mathbf{g}(p)$, then the program can derive the following:

$\mathbf{g}(p)$: The user can view the summary of her shopping cart at any time during her use of the web application.

The program deduces the goal $\mathbf{g}(p)$, and we can informally interpret this as saying that the database contains information about how to satisfy the goal $\mathbf{g}(p)$. Knowing this, we could choose that our next task be to update Δ to ensure that it satisfies another goal.

The program allows us to distinguish between two sets of information: the requirements database and the information derivable from the requirements database, and used to inform the design and identification of candidate solutions to the requirements problem that the database defines. Design, because given goals and quality constraints, we need to define ways of satisfying them; identification, because there will be different ways to satisfy goals and quality constraints, and some combinations of these ways will be feasible and will be called candidate solutions to the requirements problem, others not.

By distinguishing the database from the oracle, the design and identification of candidate solutions can proceed through an interactive process: after every change to the requirements database, consult the oracle to see – through its pronouncement – what changed in the candidate solutions, and use that feedback to decide what to change next. The better the program tolerates deficient inputs and the more informative the oracle, the more relevant this process is.

It may seem at this point that all an oracle pronouncement should be is the set of all logical consequences of a requirements database. This is *not* the case, mainly because the requirements database can be *inconsistent* and/or *incomplete*.

Consider inconsistency first. Suppose we find out that the user should not be able to change the contents of her shopping cart after she has started the checkout. This conflicts with $\mathbf{t}(p_1)$, as the system should not display the link to the shopping cart on the screens for the checkout. To record this into Δ, we add $\mathbf{g}(p_3)$ and $\mathbf{k}(p_4)$ to Δ.

$\mathbf{g}(p_3)$: The user should not be able to change the contents of her shopping cart after she has started the checkout.

$\mathbf{k}(p_4)$: It is not possible to show the link to the shopping cart on every screen and to ensure that the user cannot change the contents of her shopping cart during checkout. That is: $\mathbf{k}(\mathbf{t}(p_1) \wedge \mathbf{g}(p_3) \rightarrow \perp)$.

The database $\Delta' = \Delta \cup \{\mathbf{k}(p_3), \mathbf{k}(p_4)\}$ is inconsistent. What should the program deduce from the inconsistent Δ', i.e., what should be in the oracle of Δ'? Clearly, a program that allows an inconsistent requirements database as its input is more interesting than one which works only on consistent requirements databases. The latter requires us to decide how to resolve every inconsistency before consulting the oracle, which obviously reduces the benefit of working with an oracle in the first place: the oracle should inform our decisions to resolve inconsistencies.

Consider incompleteness. Suppose that we have a requirements database $\Delta'' = \{\mathbf{g}(p_3)\}$. Should we let $\mathbf{g}(p_3)$ in the pronouncement of the oracle? If we see the oracle's pronouncement as the set of all logical consequences, and if we – as many do (e.g., [1], [2], [9], [10], [8], [3]) – use classical logic to compute these consequences, then the pronouncement should include $\mathbf{g}(p_3)$, since the classical consequence relation $\vdash$ is reflexive. This, however, leads to misleading conclusions: $\mathbf{g}(p_3)$ in the pronouncement would mean that $\mathbf{g}(p_3)$ is satisfied, yet there are no tasks related to that goal in Δ''. The requirements database is incomplete: while $\mathbf{g}(p_3)$ is not involved in inconsistencies, this is not evidence enough that it is satisfied, since the database says nothing on *how* to satisfy it.

The program that computes oracles should tolerate both inconsistency and incompleteness of requirements databases. The resulting oracles should help us find inconsistencies that need to be resolved, as well as ways in which we can improve the completeness of the requirements database.

The purpose of this paper is to define a set of desirable properties of pronouncements of requirements oracles, and thereby characterize the information that we want to find in pronouncements of consistent or inconsistent, and complete or incomplete requirements databases. The properties can be understood as general requirements that any decision support system should satisfy in order to assist the requirements engineer in the design of solutions to requirements problems.

The definition of properties comes out of, and renders precise intuitions about the role of requirements databases in requirements engineering (hereafter RE). Two intuitive ideas act, in this paper, as principles that guide the definition of the properties of oracle pronouncements:

1) *Pronouncement on inconsistent requirements databases should constructively tolerate inconsistency.* Two ways of not being constructive (i.e., being unhelpful) must be avoided. Firstly, if, in case Δ is inconsistent, its oracle's pronouncement $\Omega(\Delta)$ is empty, then this is useful only to the extent that it signals that Δ is inconsistent. It is not constructive, since not everything in Δ may participate in the inconsistency, and it would be interesting to know what is not involved in the inconsistency. The second obvious way to be unconstructive is to allow every expression in $\Omega(\Delta)$ when Δ is inconsistent, and thereby follow the *ex falso quodlibet* proof rule, that anything can be deduced from an inconsistency. Such a case, again, tells us only that there is an inconsistency in Δ.
2) *Ontological classification of some $\phi \in \Delta$ and the relations in which ϕ stands to other members of Δ should influence whether ϕ is in $\Omega(\Delta)$.* E.g., some information in the requirements base will state goals to achieve, others domain assumptions to maintain, and some will be tasks to execute in order to satisfy goals. These kinds of information are specified in an ontology for requirements, and that ontology should be taken into account when defining what goes in and stays out of a pronouncement of a requirements oracle.

It will become clear throughout the paper exactly how these principles influence the choices of the properties of requirements oracles.

The rest of the paper is organized as follows. We first define the concepts and relations used for the classification of requirements in the requirements database (§II). We then present the formalism in which we define the properties (§III) and use the formalism to define the requirements problem and candidate solution concepts (§IV). The properties are then defined (§V). We overview related work (§VI), and close the paper with a discussion of contributions and of open issues (§VII).

II. Ontology

The ontology for requirements performs the classification of bits and pieces of information elicited or otherwise obtained in the early steps of RE, and stored in the requirements database. The most abstract concepts and relations used below come from the core ontology for requirements [5], [6], others are specializations thereof.

A. Concepts

Each piece of information that is collected is taken to be an *expression*, and is represented by a pair of the form

$$(\textit{instantiated concept symbol}, \textit{expression symbol}). \quad \text{(II.1)}$$

The *expression symbol* refers to the actual piece of information, and the *instantiated concept symbol* refers to the concept from the ontology, which is instantiated by the expression.

The ontology for requirements defines the set of concepts that are instantiated in order to categorize the collected information. The set of concepts comes with a set of symbols, one for each concept, and used to label expressions. Below are the informal definitions of the concepts, with the corresponding labels on the left-hand side.

k: **Domain assumptions:** an expression is an instance of the Domain assumption concept if it refers to conditions that are believed to hold.

g: **Goals:** an expression is an instance of the Goal concept if it refers to conditions, the satisfaction of which is desired, binary, and verifiable (e.g., desired functionalities of the system-to-be).

q: **Quality constraints:** an expression is an instance of the Quality constraint concept if it refers to conditions that constrain the desired values of non-binary measurable properties or behaviors (e.g., the number of users that can be logged on concurrently, the response time of servers).

- **s**: **Softgoals:** an expression is an instance of the Softgoal concept if it refers to a vague condition that constrains desired values of (potentially) not directly measurable properties or behaviors (e.g., software should respond quickly, interface should be usable).
- **t**: **Tasks:** an expression is an instance of the Task concept if it refers to behaviors of the system-to-be and/or within its operating environment (e.g., find, transform, produce data, manipulate objects).

Given an expression, we use the definitions of the concepts to categorize that expression, and associate a label to it to make its categorization explicit. The ontology thereby gives a total function, called the sorting function.

Definition II.1. ***Sorting function*** *is the total function:*

$$\mathit{Sort} : \mathcal{L} \longrightarrow \mathcal{O}, \tag{II.2}$$

where $\mathcal{L}$ is the set of all instances of the Domain assumption, Goal, Quality constraint, Softgoal, Task concepts; and

$$\mathcal{O} \overset{def}{=} \{\boldsymbol{k}, \boldsymbol{g}, \boldsymbol{q}, \boldsymbol{s}, \boldsymbol{t}\} \tag{II.3}$$

is the set of labels, each corresponding to a concept in the ontology.

Remark II.2. We use lowercase Greek letters to denote individual labeled expressions. Uppercase Greek letters denote sets of labeled expressions. We index or prime these symbols as needed. When we want to emphasize the label of an expression (e.g., ϕ), we write the label along with the symbol of the expression (e.g., if $\mathit{Sort}(\phi) = \mathbf{g}$, then we write $\mathbf{g}(\phi)$). ■

B. Relations

We use six relations between expressions:

- $\wedge$ denotes the binary conjunction relation.
- $\rightarrow$ denotes the implication relation, taken here to refer to the conditional relation, i.e., $\phi \rightarrow \psi$ abbreviates "if ϕ then ψ".
- $\rightarrow \perp$ denotes the n-ary conflict relation, where the left-hand side of the implication is a conjunction. The conflict relation of the form $\phi \wedge \psi \rightarrow \perp$ abbreviates "ϕ and ψ cannot be satisfied together."
- $\succ$ denotes the binary and irreflexive preference relation, such that if $\phi \succ \psi$, then not $\psi \succ \phi$. $\phi \succ \psi$ states that ensuring that ϕ holds is strictly more desirable than ensuring that ψ holds.
- ϕ^{M} denotes the unary is-mandatory relation which indicates that ϕ *must* be satisfied.
- ϕ^{O} denotes the unary is-optional relation which indicates that it is more desirable to satisfy ϕ than to violate it, but that violating it is allowed.

The is-optional and is-mandatory relations allow us to specialize every concept onto three concepts. For example, we distinguish three kinds of domain assumptions:

- $\mathbf{k}^{\mathsf{M}}$: **Strict domain assumptions:** an expression is an instance of Strict domain assumption if it is an instance of Domain assumption and is mandatory, i.e., it must be satisfied (e.g., laws of physics, legal norms).
- $\mathbf{k}^{\mathsf{O}}$: **Defeasible domain assumptions:** an expression is an instance of Defeasible domain assumption if it is an instance of Domain assumption and is optional, i.e., it can be violated, but it is more desirable to satisfy it than to violate it (e.g., assumptions about the availability of resources).
- **k**: **Ordinary domain assumptions** an expression is an instance of Ordinary domain assumption if it is an instance of Domain assumption and is neither optional nor mandatory, i.e., it refers to conditions that can be violated, and we are indifferent if these conditions are violated or satisfied.

Remark II.3. Why is the class of every concept *not* partitioned onto only optional and mandatory instances? I.e., why are some instances called ordinary and thereby are neither mandatory, nor optional? We need the ordinary instances for cases when there is indecision about an instance's mandatory or optional status, or if there is no need to make that decision. We have the former case, indecision, when we simply choose not to have either of these two relations on an instance. The latter case, when the decision is unnecessary, occurs for so-called intermediary instances: e.g., if we say that to satisfy a goal $\mathbf{g}(\phi)^{\mathsf{M}}$, we could execute either task $\mathbf{t}(\psi_1)$ or task $\mathbf{t}(\psi_2)$, but that we cannot do both of these tasks (i.e., they are in conflict), then we should state that one of these tasks is mandatory *only if* we want to ensure that the other task is not executed. If we do not want to exclude alternatives in this way, then there is no need to decide on which of the alternative tasks must be executed, and thus the two tasks are neither mandatory nor optional. ■

Each concept referred to by symbols in $\mathcal{O}$ can be specialized along the same approach onto its instances which are mandatory, optional, or ordinary.

The expressions stating the is-optional, is-mandatory, preference, and quantification relations on members of $\mathcal{L}$ are themselves outside $\mathcal{L}$. We further distinguish three sets in $\mathcal{L}$: $\mathcal{L}^{a}$ is the set of *atomic facts* in which no expression states a relation between atomic facts, $\mathcal{L}^{\rightarrow}$ in which every expression gives conjunction and implication relations (but not the conflict relation) on atomic facts, $\mathcal{L}^{\perp}$ where every expression states a conflict relation on atomic facts. The full language is thus:

$$\mathcal{L}^{f} \overset{def}{=} \mathcal{L}^{a} \cup \mathcal{L}^{\rightarrow} \cup \mathcal{L}^{\perp} \cup \mathcal{L}^{\succ} \cup \mathcal{L}^{\mathsf{M}} \cup \mathcal{L}^{\mathsf{O}}, \tag{II.4}$$

where every member of $\mathcal{L}^{\mathsf{M}}$ and $\mathcal{L}^{\mathsf{O}}$ states, respectively, an is-mandatory or is-optional relation on an expression in $\mathcal{L}$, and every member of $\mathcal{L}^{\succ}$ a preference relation between two members of $\mathcal{L}$. We partition the language onto the sets in Equation II.4, so that

$$\forall X, Y \in \{\mathcal{L}^{a}, \mathcal{L}^{\rightarrow}, \mathcal{L}^{\perp}, \mathcal{L}^{\succ}, \mathcal{L}^{\mathsf{M}}, \mathcal{L}^{\mathsf{O}}\},\ X \cap Y = \emptyset, \quad \text{(II.5)}$$

with $\mathcal{L} = \mathcal{L}^{a} \cup \mathcal{L}^{\rightarrow} \cup \mathcal{L}^{\perp}$.

III. Formalism

We assume that the information elicited or otherwise acquired during RE is stored in a requirements database. We view this database as a subset of the language, $\Delta^{f} \subseteq \mathcal{L}^{f}$.

Definition III.1. ***A requirements database****, denoted Δ^{f}, is a set of expressions from $\mathcal{L}^{f}$: $\Delta^{f} \subseteq \mathcal{L}^{f}$.*

We restrict the language only to the expressions in $\mathcal{L}$, and we are thus interested only in the part of the requirements database $\Delta \subset \Delta^{f}$ such that $\Delta \subseteq \mathcal{L}$.

Definition III.2. ***The language*** *$\mathcal{L}$ is a finite nonempty set of expressions, in which every expression $\phi \in \mathcal{L}$ satisfies the following* BNF *specification:*

$$a ::= \boldsymbol{k}(p) \mid \boldsymbol{g}(p) \mid \boldsymbol{q}(p) \mid \boldsymbol{s}(p) \mid \boldsymbol{t}(p) \quad \text{(III.6)}$$

$$b ::= a \mid a^{\mathsf{O}} \mid a^{\mathsf{M}} \quad \text{(III.7)}$$

$$c ::= \left(\bigwedge_{i=1}^{n \geq 1} b_i\right) \rightarrow b \mid \left(\bigwedge_{i=1}^{n \geq 2} b_i\right) \rightarrow \perp \quad \text{(III.8)}$$

$$\phi ::= b \mid \boldsymbol{k}(c) \mid \boldsymbol{k}(c)^{\mathsf{O}} \mid \boldsymbol{k}(c)^{\mathsf{M}}, \quad \text{(III.9)}$$

where p in a is a proposition, and every labeled proposition, i.e., every a generated by the BNF *specification above is a member of $\mathcal{L}^{a}$ and is called an atomic fact.*

We define the consequence relation for the fragment $\mathcal{L}$ of $\mathcal{L}^{f}$ as follows.

Definition III.3. ***Consequence relation*** *$\mid\!\sim_{\tau}$ is such that, for $\Pi \subseteq \mathcal{L}$, $\phi \in \mathcal{L}$ and $x \in \{\phi, \perp\}$:*

- *$\Pi \mid\!\sim_{\tau} \phi$ if $\phi \in \Pi$, or*
- *$\Pi \mid\!\sim_{\tau} x$ if $\forall 1 \leq n$, $\Pi \mid\!\sim_{\tau} \phi_i$ and $\boldsymbol{k}(\bigwedge_{i=1}^{n} \phi_i \rightarrow x)^{\boldsymbol{y}} \in \Pi$, for $\boldsymbol{y} \in \{$"empty"$, \mathsf{O}, \mathsf{M}\}$.*

Remark III.4. The consequence relation $\mid\!\sim_{\tau}$ is sound with regards to standard entailment $\vdash$ in classical propositional logic, but is incomplete in two ways: it only considers deducing positive atomic facts, and no ordinary proofs based on arguing by contradiction go through, thus being paraconsistent. ■

We use specific sets and the operationalization function in the definition of the Requirements problem and Candidate solution concepts.

Definition III.5. ***Useful sets:*** *Let $\Pi \subseteq \mathcal{L}$, we define:*

$$\mathit{CON}(\Pi) = \{\Phi \subseteq \Pi \mid \Phi \not\mid\!\sim_{\tau} \perp\} \quad \text{(III.10)}$$

$$\mathit{INC}(\Pi) = \{\Phi \subseteq \Pi \mid \Phi \mid\!\sim_{\tau} \perp\} \quad \text{(III.11)}$$

$$\mathit{MC}(\Pi) = \{\Phi \in \mathit{CON}(\Pi) \mid \forall \Psi \in \mathit{CON}(\Pi),\ \Phi \not\subset \Psi\} \quad \text{(III.12)}$$

$$\mathit{MI}(\Pi) = \{\Phi \in \mathit{INC}(\Pi) \mid \forall \Psi \in \mathit{CON}(\Pi),\ \Psi \not\subset \Phi\} \quad \text{(III.13)}$$

$$\mathit{Rules}(\Pi) = \Pi \cap (\mathcal{L}^{\rightarrow} \cup \mathcal{L}^{\perp}) \quad \text{(III.14)}$$

$$\mathit{AtF}(\Pi) = \Pi \cap \mathcal{L}^{a} \quad \text{(III.15)}$$

$$\mathit{MI}_{\boldsymbol{x}}^{\boldsymbol{y}}(\Pi) = \{\Phi \in \mathit{MI}(\Pi) \mid \forall \phi \in \mathit{AtF}(\Phi),\quad \boldsymbol{x}\phi^{\boldsymbol{y}}\} \quad \text{(III.16)}$$

$$\mathit{AtF}_{\boldsymbol{x}}^{\boldsymbol{y}}(\Pi) = \{\phi \in \mathit{AtF}(\Pi) \mid \boldsymbol{x}\phi^{\boldsymbol{y}}\} \quad \text{(III.17)}$$

$$X^{\boldsymbol{y}}(\Pi) = \{\phi \in \Pi \mid \boldsymbol{x}\phi^{\boldsymbol{y}}\} \quad \text{(III.18)}$$

where $\boldsymbol{x} \in \{\boldsymbol{k}, \boldsymbol{g}, \boldsymbol{q}, \boldsymbol{s}, \boldsymbol{t}\}$ and $\boldsymbol{y} \in \{$"empty"$, \mathsf{O}, \mathsf{M}\}$.

$\mathit{CON}(\Pi)$ is the set of all consistent subsets of Π; $\mathit{INC}(\Pi)$ is the set of all inconsistent subsets of Π; $\mathit{MC}(\Pi)$ is the set of all maximally consistent subsets of Π; and $\mathit{MI}(\Pi)$ is the set of all minimally inconsistent subsets of Π.

Every member of $\mathit{MI}_{\boldsymbol{x}}^{\boldsymbol{y}}(\Pi)$ is a minimally inconsistent set in which all atomic facts are instances of a particular concept: e.g., $\mathit{MI}_{\boldsymbol{g}}(\Pi)$ gives all minimally inconsistent sets in which every atom is a goal (be it mandatory, optional, or neither), while $\mathit{MI}_{\boldsymbol{g}}^{\mathsf{M}}(\Pi)$ gives all minimally inconsistent sets in which every atomic fact is a mandatory goal.

Every member of $\mathit{AtF}_{\boldsymbol{x}}^{\boldsymbol{y}}(\Pi)$ is an atomic fact in Π that has the sort $\boldsymbol{x}$ and unary relation $\boldsymbol{y}$. E.g., $\mathit{AtF}_{\boldsymbol{g}}^{\mathsf{M}}(\Pi)$ is the set of all mandatory goals in Π.

Every member of $X^{\boldsymbol{y}}(\Pi)$ is a member of Π that has the sort $\boldsymbol{x}$ and unary relation $\boldsymbol{y}$. E.g., $\boldsymbol{K}^{\mathsf{M}}(\Pi)$ is the set of all strict domain assumptions in Π.

Definition III.6. ***The operationalization function****, for $\Delta \subseteq \mathcal{L}$, $\boldsymbol{x}, \boldsymbol{y} \in \{$"empty"$, \mathsf{O}, \mathsf{M}\}$, $\boldsymbol{Z} \in \{\boldsymbol{G}, \boldsymbol{Q}, \boldsymbol{S}\}$, and $\boldsymbol{W} \in \{\boldsymbol{K}, \boldsymbol{T}\}$:*

$$\mathit{Op} : \bigcup_{\boldsymbol{Z},\boldsymbol{x}} \boldsymbol{Z}^{\boldsymbol{x}}(\Delta) \longrightarrow \wp(\wp(\bigcup_{\boldsymbol{W},\boldsymbol{y}} \boldsymbol{W}^{\boldsymbol{y}}(\Delta))) \quad \text{(III.19)}$$

is defined as follows, for $a \in \{\boldsymbol{g}(\phi)^{\boldsymbol{x}}, \boldsymbol{q}(\phi)^{\boldsymbol{x}}, \boldsymbol{s}(\phi)^{\boldsymbol{x}}\}$:

$$\begin{aligned} \mathit{Op}(a) \overset{def}{=} \{\Pi \in \mathit{CON}(\Delta) \mid & \ \Pi \mid\!\sim_{\tau} a \\ & \text{and } \Pi \setminus \{a\} \subseteq \boldsymbol{W}^{\boldsymbol{y}}(\Delta), \\ & \text{and } \nexists \Phi \subset \Pi,\ \Phi \mid\!\sim_{\tau} z\}. \end{aligned} \quad \text{(III.20)}$$

Every member of the set $\mathit{Op}(z)$ is a minimal consistent set of tasks and domain assumptions that is sufficient to operationalize the goal, quality constraint, or softgoal a. Informally, $\mathit{Op}(z)$ tells us all ways in (i.e., subsets of) Δ of satisfying a goal, quality constraint, or softgoal.

IV. Problem and Candidate Solution Concepts

The general definition of the requirements problem differs from earlier ones as it dispenses with the stating of properties of solutions. Rather, the challenge is to find candidate solutions and compare them on the basis of is-optional, is-mandatory, and preference relations on their ingredients, so as to choose one candidate as the solution to the requirements problem. This transfers the properties of solutions into the candidate solution concept.

Definition IV.1. ***The requirements problem: Given*** *a requirements database $\Delta^f \subseteq \mathcal{L}^f$,* ***find*** *candidate solutions in Δ^f and* ***compare*** *them on the basis of preference, is-optional, and is-mandatory relations, in order to select one candidate as the solution to the requirements problem.*

Remark IV.2. The requirements problem definition does not say how to use the preference, is-optional, and is-mandatory relations to compare candidate solutions. This is intentional, as any such comparison can involve trade-offs, for which no domain- and/or project-independent rule of resolution applies. How to resolve tradeoffs, e.g., how to lead stakeholder negotiations to that aim thereby remains outside of the problem definition. ■

Definition IV.3. ***A candidate solution*** *$\mathcal{S}$ to the requirements problem given by a requirements database Δ^f is a set*

$$\mathcal{S} \subseteq \bigcup_{\boldsymbol{X},\boldsymbol{y}} \boldsymbol{X}^{\boldsymbol{y}}(\Delta),\ \boldsymbol{X} \in \{\boldsymbol{K}, \boldsymbol{T}\},\ \boldsymbol{y} \in \{\text{“empty”}, \mathsf{o}, \mathsf{M}\} \quad \text{(IV.21)}$$

of domain assumptions and tasks, which satisfies the following conditions:

1) *Consistency: $\mathcal{S} \not\mathrel{|\!\!\sim}_{\tau} \bot$;*
2) *Achievement: $\forall \phi \in \boldsymbol{Z}^{\mathsf{M}}(\Delta)$, $\boldsymbol{Z} \in \{\boldsymbol{G}, \boldsymbol{Q}, \boldsymbol{S}\}$, we have that $\mathcal{S} \mathrel{|\!\!\sim}_{\tau} \phi$;*
3) *Conformity: $\boldsymbol{K}^{\mathsf{M}}(\Delta) \cup \boldsymbol{T}^{\mathsf{M}}(\Delta) \subseteq \mathcal{S}$;*
4) *Dominance: $\nexists \mathcal{S}'$ such that $\mathcal{S}'$ is a candidate solution and $\exists \boldsymbol{X}^{\mathsf{o}}(\Delta) = \mathcal{S}' \setminus \mathcal{S}$, such that $\boldsymbol{X}^{\mathsf{o}}(\Delta) \neq \emptyset$ and $\boldsymbol{X} \in \{\boldsymbol{K}, \boldsymbol{T}\}$;*
5) *Minimality: $\nexists \mathcal{S}'$ such that $\mathcal{S}'$ is a candidate solution and $\mathcal{S}' \subset \mathcal{S}$.*

The Achievement condition requires that a candidate solution satisfies all mandatory goals, quality constraints, and softgoals. Satisfaction is formalized by entailment, so that we assume a goal, quality constraint, or softgoal is satisfied if we can derive it from $\mathcal{S}$. It is not difficult to see, from Definitions III.6 and IV.3, that $\mathcal{S}$ is required to include an operationalization of every mandatory goal, quality constraint, and softgoal.

The Conformity condition asks that all strict domain assumptions are not violated and all mandatory tasks are executed. The Achievement and Conformity conditions ensure that the candidate solution satisfies all that *must* be satisfied.

According to the Dominance condition, every candidate solution will be maximal with regards to optional requirements. This condition formalizes the idea of the is-optional relation, as holding on requirements which are desirable to satisfy, but can be violated. A candidate solution will thus include as many defeasible domain assumptions and as many optional tasks, up to the point at which adding any further defeasible domain assumptions and/or optional tasks violates the Consistency, Achievement, Conformity, or Minimality conditions.

The Minimality condition requires that a candidate solution includes only the domain assumptions and tasks which are needed to satisfy exactly the Consistency, Achievement, Conformity, and Dominance conditions.

V. Properties of Oracle Pronouncements

The purpose of oracle pronouncements is to help us identify candidate solutions starting from an incomplete and/or inconsistent requirements database. The properties should therefore ensure that there is an oracle pronouncement for every candidate solution. However, since an oracle pronouncement can be such that it can become a candidate solution only by further updates of the requirements database, not every pronouncement will include a candidate solution.

Remark V.1. We denote Ω^{I} an oracle in this first case of basic properties, while $\Omega^{\mathsf{I}}(\Delta)$ denotes the set of pronouncements of that oracle, given the requirements database Δ. ■

Definition V.2. ***An oracle pronouncement*** *is a set of formulas $P \subseteq \mathcal{L}$ which satisfies the properties* ***Con****,* ***Cl****,* ***M****,* ***KM****,* ***TM****,* ***KD****, and* ***TO****.*

As we require the pronouncement to include a potential candidate solution, it should be consistent.

Con: $\forall P \in \Omega^{\mathsf{I}}(\Delta),\ P \not\mathrel{|\!\!\sim}_{\tau} \bot.$

The pronouncement should say if its members satisfy goals, quality constraints, or softgoals, and more generally, include all logical consequences of the requirements that it includes. Such a pronouncement helps us decide,m e.g., which goals to further refine in the requirements database, when these goals are absent from all pronouncements.

Cl: $\forall P \in \Omega^{\mathsf{I}}(\Delta),\ P = \{\phi \mid P \mathrel{|\!\!\sim}_{\tau} \phi\}.$

The is-mandatory relation states that a requirement it applies to must be satisfied. It follows that, if there are inconsistencies between only between mandatory requirements, the oracle should remain silent, until these inconsistencies have been resolved.

M: $\Omega^{\mathsf{I}}(\Delta) = \emptyset$ if $\exists \Pi \subseteq \mathsf{MI}(\Delta)$ s.t. $\Pi \subseteq \mathbf{X}^{\mathsf{M}}(\Delta)$,
where $\mathbf{X} \in \{\mathbf{K}, \mathbf{G}, \mathbf{Q}, \mathbf{S}, \mathbf{T}\}$.

All pronouncements should include mandatory domain assumptions and tasks, since every candidate solution should include them.

KM: $\forall P \in \Omega^{\mathsf{I}}(\Delta),\ \mathbf{K}^{\mathsf{M}}(\Delta) \subseteq P.$

TM: $\forall P \in \Omega^{\mathsf{I}}(\Delta),\ \mathbf{T}^{\mathsf{M}}(\Delta) \subseteq P.$

Pronouncements which include as many as possible of the defeasible domain assumptions from Δ are the most interesting.

KD: $\forall P \in \Omega^{!}(\Delta)$, $\nexists P' \in \Omega^{!}(\Delta)$ s.t. $P' \setminus P = \mathbf{K^o}(\Delta)$ and $\mathbf{K^o}(\Delta) \neq \emptyset$.

Same applies to pronouncements in relation to optional tasks.

TO: $\forall P \in \Omega^{!}(\Delta)$, $\nexists P' \in \Omega^{!}(\Delta)$ s.t. $P' \setminus P = \mathbf{T^o}(\Delta)$ and $\mathbf{T^o}(\Delta) \neq \emptyset$.

We want pronouncements to include all goals, quality constraints, and softgoals that they operationalize.

G: $\forall P \in \Omega^{!}(\Delta)$, if $\exists \Pi \in \mathsf{Op}(\mathbf{g}(\phi)^{\mathbf{x}})$ s.t. $\mathbf{x} \in \{\text{"}empty\text{"}, \textsc{o}, \textsc{m}\}$ and $\Pi \subseteq P$, then $\mathbf{g}(\phi)^{\mathbf{x}} \in P$.

Q: $\forall P \in \Omega^{!}(\Delta)$, if $\exists \Pi \in \mathsf{Op}(\mathbf{q}(\phi)^{\mathbf{x}})$ s.t. $\mathbf{x} \in \{\text{"}empty\text{"}, \textsc{o}, \textsc{m}\}$ and $\Pi \subseteq P$, then $\mathbf{q}(\phi)^{\mathbf{x}} \in P$.

S: $\forall P \in \Omega^{!}(\Delta)$, if $\exists \Pi \in \mathsf{Op}(\mathbf{s}(\phi)^{\mathbf{x}})$ s.t. $\mathbf{x} \in \{\text{"}empty\text{"}, \textsc{o}, \textsc{m}\}$ and $\Pi \subseteq P$, then $\mathbf{s}(\phi)^{\mathbf{x}} \in P$.

Proposition V.3. *Properties **G**, **Q**, and **S** are derived from properties **Con** and **Cl**.*

Proof: Obvious, from Definition III.6. ■

The following result relates oracle pronouncements to candidate solutions.

Theorem V.4. *For every candidate solution $\mathcal{S}$ of the requirements problem stated by the requirements database Δ, there is an oracle pronouncement $P \in \Omega^{!}(\Delta)$ such that $\mathcal{S} \subseteq P$.*

Proof: (By contradiction.) Assume the opposite, that there is a candidate solution $\mathcal{S}$ of the requirements problem stated by the requirements database Δ, such that $\forall P \in \Omega^{!}(\Delta)$, $\mathcal{S} \nsubseteq P$.

For the assumption to hold, every $P \in \Omega^{!}(\Delta)$ should violate at least one of the conditions other than Minimality in the definition of the candidate solution concept (cf., Definition IV.3). We consider the consequences of this being the case:

1) If P violates Consistency, then it violates **Con** and is not a pronouncement of the oracle, which contradicts the assumption that $P \in \Omega^{!}(\Delta)$.
2) If P violates Achievement, then there is a mandatory goal, quality constraint, and/or softgoal which is not in P. This can be the case only if P does not operationalize that goal, quality constraint, and/or softgoal. If every $P \in \Omega^{!}(\Delta)$ violates Achievement, then there are mandatory goals, quality constraints, and softgoals in Δ which are not satisfied, and there are therefore no candidate solutions in Δ.
3) If P violates Conformity, it violates **KM** or **TM**, which contradicts that $P \in \Omega^{!}(\Delta)$.
4) If P violates Dominance, then it violates **KD** or **TO**, which contradicts that $P \in \Omega^{!}(\Delta)$.
5) If P violates Minimality, then P can still include a candidate solution.

■

Theorem V.5. *There can exist $P \in \Omega^{!}(\Delta)$ such that there is no $\mathcal{S} \subseteq P$ and that $\mathcal{S}$ is also a candidate solution of the requirements problem stated by the requirements database Δ.*

Proof: There is no property which requires that every $P \in \Omega^{!}(\Delta)$ includes all mandatory goals. An $P \in \Omega^{!}(\Delta)$ which does not include all mandatory goals violates Achivement, yet is still an oracle pronouncement. ■

VI. Related Work

From the theoretical standpoint, this is a paper about properties of knowledge bases that are useful in requirements engineering. The ontology provides the classification of information in a database, and the oracle processes the database to filter out what is useful to the requirements engineer. The properties formalize the ideas stated by the ontology for requirements, thereby giving a clear role to the ontology, beyond the usual notion of using it to organize a specification. We are not aware of work that has had this same aim, or that has offered an analysis of desirable properties of knowledge bases for the resolution of the requirements problem.

From the engineering perspective, the properties of oracle pronouncements are themselves requirements that need to be satisfied by tools for requirements representation and analysis. Tool-support for RE has received considerable attention. Software has been developed to support the modeling activities, as well as various analysis activities. The survey of goal-oriented requirements engineering, from Kavakli & Loucopoulos [7] points to various tools available for modeling goals and reasoning about goal models, including, e.g., Objectiver [11] and T-Tool [3]. Tools support animation, consistency checks, possibility checks (to check for over-specification), and assertion checks (to check for under-specification). Apart from the idea that the tool should help us check if one or more goals are satisfied, an ontology for requirements does not relate in other ways to the use of the tool. The properties introduced here suggest how the ontology suggests questions to ask from the tool.

VII. Discussion and Conclusions

This paper continues our previous work towards a general definition of the requirements problem through an ontology of core concepts in RE [5], [6], and the subsequent definition of a simple but abstract formalism, Techne, for the representation of requirements problems and candidate solutions [4] during the early steps of RE. In this paper, we show how candidate solutions satisfy some normally unrelated, but intuitively appealing properties which are often adopted by default in RE research: e.g., that the requirements specification of a system design must operationalize goals, that the specification must say how to satisfy all mandatory requirements, and so on. Formalization of the properties gives an idea of how they come together and relate to the candidate solutions sought in relation to a requirements problem.

References

[1] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[2] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Trans. Softw. Eng.*, 20(8):569–578, 1994.

[3] A. Fuxman, L. Liu, J. Mylopoulos, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Eng.*, 9(2):132–150, 2004.

[4] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos. Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling. In *18th IEEE Int. Requirements Eng. Conf.*, 2010.

[5] I. J. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *16th IEEE Int. Requirements Eng. Conf.*, pages 71–80, 2008.

[6] I. J. Jureta, J. Mylopoulos, and S. Faulkner. A core ontology for requirements. *Applied Ontology*, 4(3-4):169–244, 2009.

[7] E. Kavakli and P. Loucopoulos. Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods. In J. Krogstie, T. Halpin, and K. Siau, editors, *Information Modeling Methods and Methodologies*. IGI, 2005.

[8] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements interaction management. *ACM Comput. Surv.*, 35(2):132–190, 2003.

[9] A. van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Trans. Software Eng.*, 24(11):908–926, 1998.

[10] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Software Eng.*, 26(10):978–1005, 2000.

[11] Axel van Lamsweerde. Goal-oriented requirements enginering: A roundtrip from research to practice. In *RE*, pages 4–7. IEEE Computer Society, 2004.