Formal validation and requirements management based on the Jackson's reference model for requirements and specifications

Takashi KITAMURA*, Keishi OKAMOTO†, Makoto TAKEYAMA‡
Collaborative Research Team for Verification, National Institute of Advanced Industrial Science and Technology (AIST)
*Email: * t.kitamura@aist.go.jp † keishi.okamoto@aist.go.jp ‡ makoto.takeyama@aist.go.jp*

***Keywords*- Formal methods, Jackson's problem frame**

I. INTRODUCTION

This research aims to develop a formal framework for (1) formal validation for satisfiability of specifications to requirements, and (2) requirements management based on the Jackson's reference model for requirements and specifications, which provides an insight and perspective basis for relationship between requirements and specifications. To develop the framework, we use propositional logic, from which we derive formal discussion and devices for computer assistance. In the framework the validation for satisfiability of specifications to requirements is ascribed to the validity checking of logical formulas. Also within the framework we develop a useful notion of "weakest adequate specifications" with its calculating technique. We will demonstrate the usefulness of the framework with practical examples.

II. JACKSON'S REFERENCE MODEL

We review the reference model for requirements and specifications proposed by Jackson et al (JRM)[2], [3], [4].

Fig. 1 depicts the key concept of the JRM. It consists of the following five artifacts which classified into two groups that pertain to the *system* and to the *environment*: *Domain knowledge* (D) which provides presumed environment facts; *Requirements* (R) that indicates what the customer needs from the system, described via its effect on the environment; *Specifications* (S) that provides information to build a system that satisfies the requirements; *Program* (P); and *Programming platform* (M).

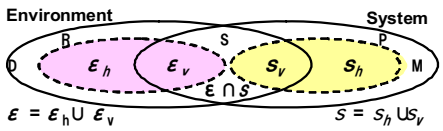

Figure 1. Five software artifacts with designated terms of the JRM.

The *designated terminology* provides classified names to represent phenomena appearing in a development. Some of the phenomena appearing in the development belong to the environment; we denote this set $\mathcal{E}$. Others belong to the system; we denote this set $\mathcal{S}$. At the interface between the environment and the system, some of the $\mathcal{E}$ phenomena are visible to the system: we denote this subset $\mathcal{E}_v$. Its complement in $\mathcal{E}$ are hidden from the system: we denote this set by $\mathcal{E}_h$. Thus $\mathcal{E} = \mathcal{E}_h \cup \mathcal{E}_v$. The $\mathcal{S}$ phenomena are similarly decomposed into $\mathcal{S}_v$ and $\mathcal{S}_h$. Terms denoting phenomena in $\mathcal{E}_h, \mathcal{E}_v$, and $\mathcal{E}_v$ are visible to the environment and used in R and D. Terms denoting phenomena in $\mathcal{S}_h, \mathcal{S}_v$ and $\mathcal{E}_v$ are visible to the system and used in P and M. Hence, only $\mathcal{E}_v$ and $\mathcal{S}_v$ are visible to both the environment and the system; we restrict specification (S) to using only these terms.

The *adequacy property* "$D \wedge S \Rightarrow R$", a relationship that should stand between requirements, specifications and domain knowledge (dk), is a main concept in the JRM.

III. FORMAL FRAMEWORK BASED ON JRM

Description languages: We prepare simple description languages for requirements, specifications and dk, based on propositional logic with arrangement to suit JRM. First we capture the notion of the *designated terms* in JRM; we presume the finite set of *propositional variables for visible phenomena* $\mathcal{A}_v$, ranged over $p^v, q^v, \cdots$ and of *variables for hidden phenomena* $\mathcal{A}_h$, ranged over $p^h, q^h, \cdots$. The description languages for requirements and dk are the propositional logic whose propositional variables are those for *visible and hidden phenomena* (i.e., $\mathcal{A}_v \cup \mathcal{A}_h$). The language for specification is a propositional logic whose propositional variables are those for *visible phenomena* ($\mathcal{A}_v$).

Example 1: Consider to realize the following requirement R_1 in a development of *aeroplane control surface* as a specification [3]: "R_1: *reverse_thrust_enabled* if *moving_on_runaway*". To realize R_1 as a specification, the developers analyze its domain property, and elicit the dk as they includes the following properties: "D_1: *moving_on_runaway* iff *wheel_turning*" and "D_2: *wheel_turning* iff *wheel_pulse_on*". Due to this analysis, a specification that satisfies R_1 is designed as follows: "S_1: *reverse_thrust_enabled* if *wheel_ pulse_on*.".

We formalize this with the propositional languages designed for JRM above. First we symbolize the phenomena, appearing in the requirement, specification and dk, as: "a^v: *reversed_thrust_enabled*", "b^h: *moving_on_runaway*", "c^h: *wheels_turning*", and "d^v: *wheel_pulses_on*". Note that phenomena a and d are visible from system and belong to $\mathcal{A}_v$ as they indexed by v, and b and c are hidden and belong to $\mathcal{A}_h$ as indexed by h. Also note that the

978-0-7695-4289-8/10 $26.00 © 2010 IEEE
DOI 10.1109/PRDC.2010.42

specification description consists only of visible terms, and the requirement and dk description are of both visible and hidden terms, being faithful to JRM. Then by applying these to JRM, the following formula can be obtained:

$$\underbrace{(b^h \leftrightarrow c^h) \wedge (c^h \leftrightarrow d^v)}_{D} \wedge \underbrace{(d^v \rightarrow a^v)}_{S} \Rightarrow \underbrace{(b^h \rightarrow a^v)}_{R}.$$

From the above formula, we can see the requirements and dk description are via both hidden and visible terms, and the specification description only via visible terms. Also we can formally validate S_1 satisfies R_1 under D_1 and D_2, as this formula is valid/tautology; i.e, in our framework validity checking of satisfiability of a specification to a requirement can be ascribed to tautology checking of logical formulas.

The weakest adequate specification (was): As the example shows, generally specifications are designed under the given requirements and the presumed dk. Also, in general there are several possible specification designs that satisfy given requirements and dk. Under such a situation, the *was* is a notion to state the weakest specification among all the specification designs that satisfy given requirements.

Definition 1: A specification S is the *weakest adequate specification* for dk D and requirements R if $D \wedge S \Rightarrow R$ and if for all S' with $D \wedge S' \Rightarrow R$, we have that $S' \Rightarrow S$. □

We generalize the notion of *was*. In some cases, we have *a part of specification* or *constraints on specification* in designing specifications. An example is we have a specification, say S_0, which has been implemented in the downstream and hence we want to reuse. That is, in such a case, it is desirable to find an additional specification design together with S_0 satisfy given requirements. The following is a generalized definition of *was* taking this into account:

Definition 2: A specification S is the *weakest adequate specification* for dk D, requirements R and specification S_0 if $D \wedge S_0 \wedge S \Rightarrow R$ and if for all S' with $D \wedge S_0 \wedge S' \Rightarrow R$, we have that $S' \Rightarrow S$. □

The next theorem shows how to calculate the *was*.

Theorem 1: The was (S_w) that satisfies requirement (R) under presumed dk (D) is obtained as follows:

$$S_w = \bigwedge_{\overrightarrow{w}} (D \wedge S_0 \Rightarrow R)[\overrightarrow{w}/\overrightarrow{q^h}]$$

where $\overrightarrow{q^h}$ means the list of hidden variables appearing in D and R, and $\overrightarrow{w}$ means all the valuations for $\overrightarrow{q^h}$. □

Example 2: We demonstrate a usage of *was* for requirements change (RC) management. Suppose now a RC occurs on the requirement R above; say, it changed to the following: "R_1': *reverse_thrust_enabled* and *lift_reduction_enabled* if *moving_on_runaway*". By symbolizing the phenomenon "lift reduction enabled" as e, R_1' can be formalized as: $R_1' = b \rightarrow (a \wedge e)$. Due to the RC, it can be formally traced S_1 does not satisfy R_1' any more, as the following formula evaluated as invalid: $D_1 \wedge D_2 \wedge S_1 \Rightarrow R_1'$. Hence, we find S_1 must be modified as well to satisfy R_1'.

To do so, the developers do similar analysis as above; i.e., the following D_3 and D_4 need to be elicited as dk: "D_3: *lift_reduction_enabled* iff *spoiler_enabled* and *aileron_enabled*", where we consider *lift_reduction_enabled* is hidden, and *spoiler_enabled* and *aileron_enabled* are visible phenomena. Then by symbolizing the phenomena as "f^v: *spoiler_enabled*" and "g^v: *aileron_enabled*", the dk is formalized as: "$D_3 = e^h \leftrightarrow (f^v \wedge g^v)$".

With the analysis we could manually elaborate and design a suitable specification to satisfy R_1, but here we show the *was* assists to do so. That is, calculating the *was* for the situation (i.e. $D_1 \wedge D_2 \wedge D_3 \wedge S_1 \Rightarrow R_1'$) using Theorem 1 gives "$d^v \rightarrow (g^v \wedge h^v)$". This together with S_1 makes "$d^v \rightarrow (a^v \wedge (g^v \wedge h^v))$", which is interpreted as "*reverse_thrust_enabled* and *spoiler_enabled* or *aileron_enabled* if *wheel_pulse_on*". And this in fact is just a suitable specification design itself in this case; this means the *was* can find a suitable specification just by symbol manipulation. Though it may not be always that *was* makes just a suitable specification design itself, it at least provides a clue to find a suitable specification design and hence can assist for developers to manually design it. Also this example demonstrates only a very simple case, however, we have confirmed that the *was* works even with more complicated cases and it is more effective and helpful in such cases.

IV. Conclusion and Future work

We proposed a formal framework for satisfiability validation of specification to requirement as well as for requirements management based on JRM. The notion of *was* with its calculation technique was developed, which plays a key role in the framework. And we demonstrated the usefulness of the framework with practical examples. Currently we are developing an integrated method for RC management based on the framework by effectively combining the formal validation and *was*, and investigating its effectiveness as a case-study with a practically large-scale requirements-specifications setting. For future work, we will extend the discussion to more expressive logics such as first-order, Hoare logic, and to practical formal specification languages, such as the model-based (VDM, Z, B-methods, etc) and algebraic ones (Maude, OBJ, CASL, etc)[1].

References

[1] D. Bjørner and M. C. Henson, editors. *Logics of Specification Languages*. Springer, 2007.

[2] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3), 2000.

[3] M.Jackson. *Software Requirements and Specifications:A Lexicon of Practice, Principles and Prejudices*. ACM Press, 1995.

[4] M.Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley Professional, 2000.

[1]The specification languages have underlying logics[1].