Ontological Analysis

P. B. Ladkin
University of Bielefeld

Appeared in: *Safety Systems 14(3), May 2005*

March 16, 2005

The completeness question (CQ) is one question with a multitude of applications. It can be expressed thus:

- Have you thought of everything?
- If yes, how do you know?

Sometimes answers to CQ may be proved formally correct (does your formal design fulfil its technical requirements specification?), sometimes not (is your requirements specification adequate?). Common hazard analysis methods such as HAZOP or FMEA cannot answer CQ. Ontological Analysis, OA, is a method for developing safety requirements, devised to address CQ explicitly.

HAZOP was developed by Mond Division of ICI in 1963 and has been successfully applied in the process industries ever since. HAZOP attempts to identify systematic safety weaknesses in system design through considering the potential effects of extreme variation of system parameters beyond the intent of the design.

In [RCC99], Redmill, Chudleigh and Catmur (hereafter RCC) explain the application of HAZOP to E/E/PE-based systems. HAZOP, they emphasise, is a team activity. The HAZOP algorithm consists of the following basic procedure:

1. Select a representation of the system (logical, pictorial, etc)
2. Identify the objects (including subsystems) in the representation
3. Identify the attributes (properties) of those objects
4. Systematically consider the potential effects of extending the values of those attributes, individually or multiply, beyond design intent
5. Record any safety-related consequences from Step 4

A HAZOP performs this procedure on a variety of representations. RCC note that Step 1 is a skilled activity relying on engineering experience. They stress the importance of trying to answer CQ. But how to do so?

In computer science, it is known how to answer CQ relative to certain criteria. It is now universal in formal specification to use hierarchies of system specifications, an idea colleagues trace back to [Par72]. One formally relates more- and less-detailed specifications in such a hierarchy through a mathematical notion of *refinement*.

Formal specification and verification have achieved conspicuous industrial successes in the telecommunications industry over the last twenty years. In the mid 1990's, my group collaborated with Michael Ferguson and Jean-Charles Gre'goire at the INRS Te'le'communications Lab in Montre'al on a project to verify a mobile-phone protocol, the Radio Link Protocol RLP1. It was a common approach in those days to think of invariant properties that you wanted the communications system to satisfy, and to prove from a formal description that they were satisfied. To me, this approach suffered from lack of an answer to CQ.

We chose a different approach. Let me call it Formal Refinement, or FR:

- Write down an abstract, extremely simple description of the system. Call it UrSpec.
- Refine UrSpec (= Spec0) by adding a little more detail, and continue refining, obtaining Spec1, Spec2, Spec3, ... It is important that each refinement be a simple, even trivial, modification of its predecessor. It is less important how many specifications are generated.

- Stop refinement when one has reached a specification of the RLP1 algorithm. Call it EndSpec.
- Prove mathematically that Spec(N+1) is a refinement of SpecN, from UrSpec=Spec0 to EndSpec.

I used the TLA+ formal language and TLA logic [Lam02]. Dirk Henkel completed the hierarchy of specifications from UrSpec to EndSpec and hand-constructed the fully formal proofs of refinement in Lamport's hierarchical proof notation, over some 100pp [Hen97]. Our INRS colleagues used model-checking to validate the implementation of RLP1 against a specification informally equivalent to EndSpec.

This approach attempts to reduce the requirements CQ to essentials. Studies have shown that most unwelcome incidents with mission-critical systems involve inadequacies in the requirements specification, e.g., [Lut91]. Sometimes these inadequacies can be quite subtle, for example, those in the TCAS airborne collision-avoidance algorithms exposed by the Überlingen mid-air collision in 2002 [Lad05]. By producing a very abstract requirements specification that is close to *obviously* correct (UrSpec), and refining it to the system design (EndSpec), formally proving the refinements all the way, any CQ concerns UrSpec alone, rendering it easier to address.

FR worked well for the telecommunications example. UrSpec described a FIFO buffer in the simplest possible way:

- two simple data structures: a sequence S, and a set D of elements of the sequence
- two operations:
  - push (append a D-element to S)
  - pop (off with S's head!)

Such specifications are learnt by almost all computer science students in their first college semester. It helped that reliable, asynchronous message transmission is aptly described as a FIFO buffer. Other cases may not be so easy. The buffer describes the behavior of the channel. The channel is

abstract, and remains so: when you buy such equipment, you don't buy a channel, but rather two end pieces of equipment that transmit and receive. The trick that simplifies the requirement is logical abstraction: it is easy in logic to reify the very bit that remains abstract through the implementation. You know this approach works, because you have the refinement proofs. Such is the power of logic, and formal hierarchy.

I chose TLA+/TLA because I knew they were adequate to the purpose, having worked with them on other projects [LadLam99]. They are powerful tools, controlling the details: everything worked without any hand-waving anywhere. Even hand-proofs remained practical. But they do require expertise and mathematical ability to use fluently; my initiation took a year and some hand-holding.

I had been thinking how to adapt FR to safety-requirements specification, to answer CQ. I proposed an orderly method of safety-requirements analysis in Chapter 9 of [LadCAS01]. I called it Causal System Analysis (CSA):

1. Select a representation R.S of the system S
2. List all - objects - their properties - their relations with each other that appear in R.S. We call this a formal ontology. These define a first-order logical language L, which contains names for all the objects, their properties and their relations, in the usual way.
3. For all accidents and hazard events E of the system S which can be expressed in L,
   (a) perform a formal causal analysis (a Why-Because Analysis, or WBA) of the occurrence of E
   (b) identify countermeasures (to remove, mitigate, or reduce the likelihood of occurrence) for E
   (c) perform a WBA of the system S' incorporating those countermeasures, to verify the effectiveness of the countermeasures

There is a CQ in CSA, in the phrase "for all accidents and hazard events E... which can be expressed in L". How do we list all such E? We chose to apply HAZOP to L. So we "parametrised" CSA, if you like, by HAZOP, which I write CSA(HAZOP).

Now I can explain OA in one phrase: Do FR using CSA(HAZOP).

The advantages are:

1. Using FR renders explicit the completeness issues in representation;
2. Using a formal ontology renders explicit the completeness issues in using HAZOP guide-words;
3. HAZOP is pretty good at what it does.

There is some flexibility in OA:

1. One doesn't have to use TLA+/TLA in FR, one can use any sufficiently expressive description technique with a formal notion of refinement.
2. One doesn't have to use HAZOP in CSA. One can use CSA(FMEA) or CSA(FMECA) or CSA(Fault Trees) or CSA(whatever).
3. One doesn't even have to use WBA in CSA, one can use some other (unworthy :-) notion of causal factor instead.

OA rests, then, on three core beliefs. I believe that requirements completeness problem can be effectively addressed by using formal hierarchy; that completeness of the HAZOP guide-word process can be practically addressed by using formal ontologies; and that effective causal analysis relies on use of the Counterfactual Test, the core of WBA. If one is doing all that, I believe one is well on the way to safety-requirements heaven, or at least well out of purgatory.

I also believe in formal thoroughness. I would insist on an adequately expressive formal representation with a formal notion of refinement for FR; on explicit formal ontologies for each refinement step; on using a notion of causal factor that may be mathematically checked for correctness of its application. Safety is about things not slipping through your development cracks; informal notions leave lots of cracks around, and render them unsurveyable.

Most usable formal methods allow an informal application with practical consumption of resources, and back up the informality with formal criteria

to allow one to check, should it be necessary, that one's analysis or development is indeed correct. Let us call these "correctness obligations". Proof obligations, such as those generated in the use of SPARK Ada, are examples of correctness obligations. So are the edges in WB-Graphs constructed during a WBA, and the collection of in-edges of any nonterminal node in a WB-Graph. The discharge (proof) of correctness obligations may require heavy use of resources. Discharging them seems to be less important than explicitly formulating them in the first place. However, I believe it important that there is a sound, complete method available for proving the correctness obligations. OA allows this mixture of informal but methodical discovery with formal correctness obligations and proof method.

Applications of OA may be found in [Stu05], [Sie05]. There has been study of formal ontologies for specific industries, using UML class notation: see e.g., [KB04] and its references. Our industrial collaborators require EndSpec in UML, so UML ontologies will prove useful in OA.

References

Hen97 Dirk Henkel, Safely Sliding Windows, Research Report RVS-RR-97-05a, RVS Group, University of Bielefeld, available through www.rvs.uni-bielefeld.de –¿ Publications

KB04 Bernd Krieg-Brückner, Formal Ontologies, Proceedings of FORMS/FORMAT04, IVA Institute for Transport Safety and Automation, Technical University of Braunschweig, 2004, ISBN

Lad05 Peter B. Ladkin, Causal Analysis of the ACAS/TCAS Sociotechnical System, in Proceedings of the 9th Australian Workshop on Safety Related Programmable Systems, Brisbane, 2004, to be available through www.crpit.org.

LadCAS01 Peter B. Ladkin, Causal System Analysis: Formal Reasoning About Safety and Failure, Draft Version 2, 2001. Final version to be published by Springer-Verlag, London.

LadLam99 Peter Ladkin, Leslie Lamport, Denis Roegel, Bryan Oliver, Lazy Caching in TLA, Distributed Computing 12:151-74, 1999.

Lam02 Leslie Lamport, Specifying Systems, Addison-Wesley, 2002.

Lut91 Robyn R. Lutz, Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems, in Proceedings of the First IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, 1993. Available from www.cs.iastate.edu/ rlutz/

Par72 David L. Parnas, On the Criteria to be Used in Decomposing Systems,into Modules, Communications of the ACM 15(12): 1053-8, December 1972. Available from www.acm.org/classics/may96

RCC99 Felix Redmill, Morris Chudleigh, James Catmur, System Safety: HAZOP and Software HAZOP, John Wiley and Sons, 1999.

Stu05 Jörn Stuphorn, Investigation of the Requirements for a Mixed Time/Event-Triggered Protocol Using the Ontological Analysis Method , Diplom Thesis, RVS Group, University of Bielefeld, available from June 2005 through www.rvs.uni-bielefeld.de ⟶ Publications

Sie05 Bernd Sieker, A Procedure for Safety-Requirements Analysis for Train Dispatching Systems, Proceedings of the Fifth Bieleschweig Workshop on System Engineering, Garching bei München, April 2005, available through www.rvs.uni-bielefeld.de ⟶ Bieleschweig Workshops.

Peter Ladkin is Professor of Computer Networks and Distributed Systems at the University of Bielefeld, and a Director of Causalis Limited