Worked example of a problem from Leveson's "Safer World"

Daniel Jackson

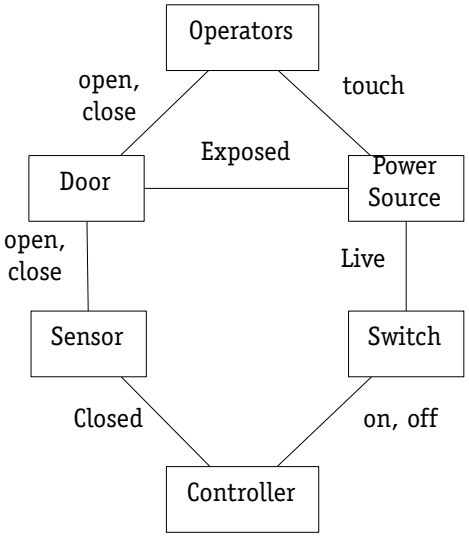
October 17, 2010

This is a worked example illustrating our approach to analyzing a system for dependability, based on a problem described by Leveson [*Engineering a Safer World*, draft at http://sunnyday.mit.edu/safer-world, pp. 180-196].

The approach starts from Michael Jackson's framework [*Problem Frames*, Addison Wesley, 2001], adapting and extending it to focus on dependability issues. [Note: we use the term 'dependability' rather than 'safety' to include systems that are critical for reasons other than safety.] This is joint research with Eunsuk Kang, Aleks Millicevic, Joseph Near and Michael Jackson.

Step 1: Context Diagram

We start with a *context diagram* showing the relevant real-world domains, the machine to be built, and the sharing of phenomena between them:



We then give *designations* for each of the shared phenomena:

open: a human operator opens the door fully or partially
close: a human operator closes the door fully
touch: a human operator touches the power source
Exposed: the power source is exposed and can be touched
Live: the power source is in a live, turned on state
Closed: the sensor is in a state that reports the door as closed
on: the controller issues a command to the switch to turn on
off: the controller issues a command to the switch to turn off

Comments.

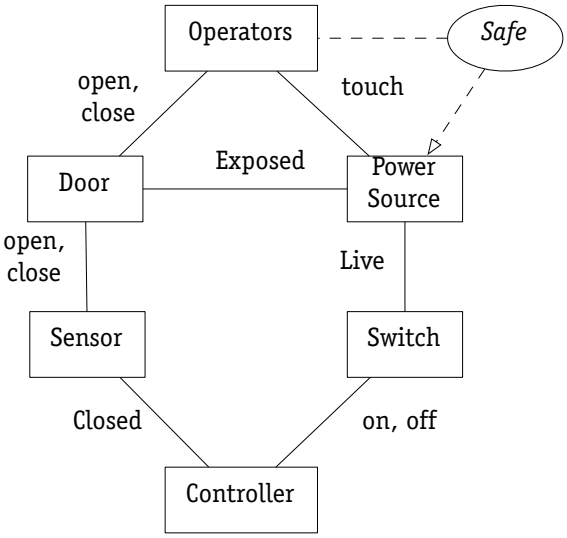
1. The context diagram bounds any subsequent analysis, so the selection of domains and phenomena must be considered very carefully. All relevant phenomena are made explicit, and are given designations giving an informal 'recognition rule' for each of the formal terms.

2. On including the domain *Operators*, we had to decide whether we would include the possibility of there being more than one operator. This seems sensible, since one can imagine scenarios in which an interlock that is designed to prevent access by a single person is overcome by two people acting collaboratively (see below). This choice is reflected in the designation for the operator events, which all mention "a human operator" rather than "the human operator".

3. Some phenomena are events and some are states. For devices (such as the Sensor and Switch) the choice is determined by their interfaces; in this case, we're assuming that the sensor provides a state that can be queried by the controller (rather than, for example, sending messages on a stream to the controller), but that the Switch is controlled by commands issued by the controller. For non-device domains, the choice is determined by what seems to give the clearest description. So, for example, the operator's behavior is described in terms of events (open the door, close the door, touch the source) rather than states (hand is not touching source, hand is touching source). States are capitalized so they can be readily distinguished from events.

Step 2: Problem Diagram

We then construct a *problem diagram* that augments the context diagram with a *requirement* showing the problem to be solved:



(The requirement is shown as an oval, with arcs to the domains whose phenomena are mentioned in the requirement. An arc with an arrow identifies a domain that will be constrained by the requirement.)

The requirement is then explicitly recorded:

> Safe: touch event does not occur in state Live

Comments.

1. In problem frames, a 'requirement' is a problem to be solved, described in terms of the real world. This is contrasted with a 'specification' which describes the intended behaviour of a component. This distinction is very important, and this clarification of terminology has been widely adopted in the requirements engineering community (see, eg, Axel van Lamsweerde, *Requirements Engineering*, 2009). The safety community seems to use the term 'requirement' in its older sense of a particular property of a component to be built (as in 'the software has 300 requirements').

2. In Leveson's approach, the notion that corresponds to our requirement seems to be the 'hazard' (in this case, 'human exposed to high energy source'). There is then a 'system safety constraint' ('the energy source must be off whenever the door is not completely closed'). From our perspective, this system safety constraint is not a requirement, but rather a specification. We want to base our analysis on the actual requirement (that no exposure occurs) and not on a proxy that might turn out to be insufficient. For example, if the operator can enter a cavity containing the power source and close the door behind him or her, the requirement will not be satisfied. It is therefore vital that the requirement itself be stated as precisely as possible, in terms of phenomena that have been designated.

3. In Michael Jackson's problem frames, the requirement usually represents the purpose of the machine being built. In our approach, the requirement is a 'critical property'. For a safety mechanism such as this, the two notions align. But in what we call 'mixed criticality systems', they may not. For example, the critical property for an online store might be that credit card information is not stolen, and one might perform an analysis to establish dependability with respect to this property alone.

Step 3: Specifications and Domain Assumptions

The next step is to associate with each domain a minimal set of domain assumptions that are believed to be needed to establish the requirement, along with a specification for each machine.

Sensor

When open occurs, Closed becomes false.

Door

Exposed is initially false.
When close occurs, Exposed becomes false.

Switch

Live is initially false.
When off occurs, Live becomes false.

Controller

Every 10ms, issue off command if Closed has become false since last check.
On command is only issued when Closed is true.

Operator

A touch event does not follow within 10ms of an open event

PowerSource

No touch event occurs unless Exposed is true.

We then record frame conditions, saying which events might affect which state components:

Only open and close affect Closed or Exposed
Only on and off affect Live

Having expressed these properties, we should now construct a logical argument that the requirement follows from the domain assumptions and specifications. Ideally, the properties are expressed formally and the argument can then be mechanically checked. Since the focus of this example is not the formal aspect, we won't pursue that further in this discussion, but an Alloy model is included with this note to illustrate how this can be done.

Comments

1. Note the clear separation between formal aspects (the statement of the properties and the analysis that they imply the requirement) from the aspects that are necessarily informal (the choice of which properties to express, the choice and designation of phenomena, etc). This is valuable because it assigns to the computer the task for which it is ideally suited, and for which human analysts perform poorly -- namely analyzing a huge space of scenarios, and assigns to the human analyst the tasks that require judgment and insight.

2. Even if a mechanical analysis is not performed, it is nevertheless valuable to make a clear

separation between the formal, logical aspects of the argument and the informal ones, in order to focus attention on the important informal aspects and ensure that there are no implicit (and unwarranted) assumptions that might go unnoticed.

3. Note that the domain assumption about the operator is by no means obvious; its validity depends on the physical layout of the door and the power source, how quickly human operators might move, whether the human operator could bypass the door, and so on. In particular, the assumption (mentioned above) of there being a single operator should be questioned, and one should consider what multiple operators might do together.

4. As soon as one starts to make the properties precise, a variety of issues arise. A particularly important one is whether phenomena are related by instantaneous constraints, or whether there are time delays. In this example, we've chosen to assume that the door sensor and switch work instantaneously, and the only delay is due to the controller's polling loop. These assumptions should of course be questioned.

5. It is best to record as *few* properties as possible, and for them to be as weak as possible, because a system whose safety relies on fewer assumptions will generally (all else being equal) be safer. Moreover, each domain assumption poses a risk, and a burden to the analyst; and the presence of irrelevant assumptions clutters the argument and makes it more likely to harbor flaws. Likewise, it is better for a specification to contain a few, weak properties even if stronger properties are expected to be satisfied, since this will reduce the verification burden and admit a wider variety of implementations. Here, for example, the specification of the controller does not mandate that the power ever be turned on, but simply demands that it *not* be turned on when the door is not closed. Perhaps more surprisingly, we have no insisted that the sensor report the door as closed when the close event occurs; the only constraint is that it report that it is *not* closed after an open event.

6. A key advantage of using a mechanical analysis is that it makes it easier to minimize assumptions. One works incrementally, starting with a minimal set of assumptions, only adding assumptions if counterexamples generated by the analysis indicate that the assumptions are not sufficient to establish the requirement. In order to do this, it is necessary to use a formalism that either supports partial constraints or (for state machine notations) allows non-determinism.

7. A mechanical analysis will in fact show that these properties are insufficient. The domain assumption for the *Operator* domain allows a close event to intervene between an open and a touch. If indeed this is not possible (for example because there is no room for the operator to reside between the door and the power source), an additional domain assumption should be added and the analysis repeated. In Leveson's analysis, this assumption is implicit, and appears only in a preamble to the analysis (which presumably justifies the later reduction of the hazard to the system safety constraint).

8. Initial conditions are very important. As Michael Jackson has noted (see *initialization concerns* in the Problem Frames book), initialization is often a tricky matter, involving synchronizing an internal model with an external reality. In this example, the condition that

the power source is initially not Live is essential. If the power source were to start live, and the door were to start in an open position, the controller would not detect a change in door position and it would remain live.

9. The design of the controller poses an interesting problem. Should the controller issue an off command whenever it reads an open state for the door, or only when a transition from closed to open is detected? One might think that if the switch is not entirely reliably, and drops (say) one in a million commands, sending redundant commands would be good. But if the switch is a primitive device that is not designed to accept a constant stream of commands, the design we chose will be preferable. However, if there is a concern that the switch may not respond reliably to the commands from the controller, it may be better to provide a feedback path -- for example, by having the controller read a sensor that gives the state of the switch or power source.

10. The specifications of machines (such as the controller) must of course be ensured, for example by verifying code.

11. In our research, we are investigating how adjustments to the design of the system might result in a smaller *trusted base* (that is, the set of domains and machines that must conform to stated properties). So should the analysis reveal a larger trusted base than is desirable, we would at this point reconsider the design and return (if necessary) to Step 1, with a new context diagram. If you look at the Alloy model, you will see that the analysis checks the requirement on the assumption that the Switch, Controller, Sensor and Door all meet their expected properties; in this example, these domains form the trusted base, and it is hard to see how it might be reduced.

12. Despite advocating recording as few properties as possible for the dependability argument, we are *not* suggesting that the system be built in this way. For example, the actual specification of the sensor should require that, in addition to the property above, a close event put it in the Closed state. Since the controller is only triggered by a transition from Closed to not Closed, a sensor that fails to meet this specification would result in the controller failing to issue an off command. However, this turns out not be a problem in our design because an on command can only be issued when the sensor reports Closed. This brittleness in the design (without the fully reliable sensor) is clearly undesirable, and poses an interesting research challenge. One could easily see whether removal of any single constraint results in a violation of the requirement (simply by commenting it out and rerunning the Alloy check). But we would expect most properties in such a pared down dependability argument to be single points of failure, and it is not clear how to identify the ones that should not be.

13. To run the analysis, simply do the following: (a) install the Alloy Analyzer; (b) open the file called *levesonInterlock.als*, and click the “Execute” button to run the check; (c) in the displayed counterexample window, select “Load Theme” from the Theme menu and select the file *levesonInterlock.thm*, which will customize the layout in a more appealing way (for example, showing the power source in red when it is live). Now just click the arrow buttons to walk through the counterexample. To fix the problem find the line in the Touch signature that has been commented (the crucial property requiring that Touch not follow Open immediately),

and remove the two hyphens. Now click “Execute” again, and no counterexamples should be found.

Step 4: Scrutinize Case for Weaknesses

We now have a ‘dependability case’ in hand, arguing that the requirement follows from the domain assumptions and machine specifications. This case should be subject to a detailed review, considering all the ways in which it might be invalid. Here is our current list of weaknesses to be investigated:

1. *Wrong domain assumptions.* Each of the domain assumptions should be carefully scrutinized, in an attempt to identify scenarios that violate the assumptions. The analyst should consider in particular whether there are phenomena that did not appear in the context diagram that might have an effect.

2. *Incorrect expression of properties.* If the domain assumptions have been formalized to allow mechanical analysis, the formalizations should be animated so that the analyst can confirm that they have been expressed correctly. In Alloy, for example, we would ask the tool to generate instances of each domain assumption and its negation, and to generate instances from combinations of assumptions; and we would also record expected implications of the assumptions and have them checked by the tool.

3. *Specifications.* The part of the case involving the satisfaction of specifications by implementations should be reviewed. In addition to the verification itself, any assumptions on which the verification relies (which should be enumerated in the case) must be reviewed. These might include, for example, the assumption that the operating system achieves address space separation; that the compiler produces faithful code; that the computer is not exposed to extremities of heat, radiation, etc; and so on.

4. *Argument analysis.* Assuming that the top-level argument has been checked mechanically, the analyst should consider ways in why the check might be unsound. The two primary risks are (a) that the tool is flawed, and (b) that an unintentional overconstraint in a domain assumption or axiom results in a vacuous proof that the requirement follows. One way to counter these risks that we commonly employ is to weaken or drop properties from domain assumptions or specifications, rerun the analysis, and check that the expected counterexamples are generated. Another strategy we use is to have the tool generate an ‘unsat core’ showing which properties were used to establish the requirement; if this is smaller than the full set of properties expected, this suggests that an overconstraint is present. Finally, taking an idea from mutation testing, one can explore the effect of making small changes to domain assumptions or specifications. If very high confidence is required, one might want to use a redundant tool to check the argument.

5. *Historical knowledge.* Most developments are what Michael Jackson calls “normal” rather than “radical” (following Vincenti). This means that many of the dependability issues will have arisen before, albeit in different contexts of use. Normal design will embody knowledge of

properties of the problem (for example, what an operator can and cannot do). The analyst will want to collect as much information as possible about historical failures, and will of course use her own judgment and experience. So, for example, the mere mention of a sensor raises questions of accuracy, delay, sensor failure, and so on. In our approach these concerns will primarily inform the articulation and assessment of the domain assumptions.

Comments on Leveson's Solution

Here are some comments on the solution that Leveson presents to the problem.

Page 181. The hazard and safety constraint are introduced. A key difference between our approaches seems to be that we regard the hazard as a property to be established explicitly within the dependability argument. Leveson focuses the analysis on the constraint, apparently leaving the relationship between hazard and constraint implicit.

The constraint says the energy source must be off "when the door is not closed". A footnote explains that "when the door is open" would be incorrect because the power controller may not have an exact model of the door position, and the case of unknown position must be accounted for. From our perspective, the top level goal should be expressed in terms of the property of concern, and not in terms of internal models, so the issue does not arise at this point. Moreover, if the sensor is not completely faithful, and the information it provides does not correspond perfectly to the actual state of the door, we would want to include not only the possibility that the state is unknown, but also that it is incorrect. Note that Leveson's comment here makes it clear that the safety constraint is about the controller, and not about the system as a whole; in our approach, the focus is on establishing system-level properties, and there is no preferred focus on the controller over other components, such as the sensors or actuators, which are equally important in ensuring safety. Finally, it should be noted that the central tenet of our approach is to make every step of the argument, and all assumptions, fully explicit; we don't think that an important issue such as this belongs in a footnote.

Page 182. The constraint is here reformulated to say that the energy source must be off whenever the door is not *completely* closed (our italics). This introduces, implicitly, another concern: namely what is meant exactly by "open" and "closed". In our approach, following Michael Jackson, this concern -- the informal interpretation of phenomena -- is fundamental, and is made fully explicit with designations.

The "functional requirements" of the power controller are not listed. These seem to be written in terms of the observable phenomena ("when the door is closed, turn off the power"). And yet Leveson's requirements are what we would call a *specification* for the controller. It seems odd that this is not expressed in terms of the interface of the controller itself. How would this approach handle multiple controllers? It seems far simpler to follow the approach advocated by Michael Jackson in which each component is given its own specification, in terms of its own interface, rather than allowing a spec for each component that talks about phenomena elsewhere in the system. Alternatively, perhaps we have misconstrued Leveson's intent here, and the phenomena being described are in fact the phenomena of the controller, so that "when

the door is closed" means "when the internal model of the controller indicates that the door is closed". This, however, does not seem to be a specification that describes observable behavior anywhere in the system. Put another way, the first requirement of the controller says "detect when the door is opened". From our point of view that cannot be the job of the controller; it is the sensor that detects the position and the controller that responds to the state of the sensor.

In the diagram, not all arcs are labelled. This again reflects the difference in emphasis on phenomena. In our approach, we want to carefully distinguish phenomena such as "the door opens" from phenomena such as "the sensor reports that the door has opened".

What exactly is the meaning of an arrow on a connection between boxes? The arc from Sensors to Power Controller marked "Door position is open…" points to the controller. So this suggests that the arrows indicate direction of data flow. But the term "control structure" suggests that these arrows show the direction of control, and if the controller polls the sensors, we would expect an arrow in the other direction.

Note that the phenomenon of an operator touching the power source is not mentioned in the diagram and is only alluded to in the text. In our approach, the critical requirement is expressed in terms of phenomena that are given formal terms and reasoned about explicitly.

When was the decision made that only one operator had to be considered? Is this relevant?

The text uses the term "automated controller" for what is called the "power controller" in the diagram. Presumably this is just a typo, since, in our view, the tendency of natural language documents to use different terms for the same things is problematic.

Page 183. The hazardous actions are now identified. The first paragraph uses the terms "safety controls" and "hazard controls"; are these the same?

Four ways in which control actions might be hazardous are listed, and a table is then generated for each case. It is not clear how this would work for a controller executing a complex state machine. Even if one considered each control action separately, its occurrence or non-occurrence might have different consequences in each state. Leveson mentions the addition of a "mode column", but doesn't explain how this will solve the problem of an explosion of cases.

A key difference between Leveson's approach and ours is the focus on the controller. We don't privilege the computer, but treat all domains equally. Leveson presumably believes that starting the analysis with the controller (or controllers in a more complex system) is beneficial; we'd like to understand why this is so. How will this approach detect, for example, the counterexample scenario in which the operator opens the door, enters the enclave, closes the door and then touches the energy source? Leveson has ruled out this scenario implicitly at the start, but in our experience these kinds of scenarios, in which unusual conditions begin with behaviors of the operator (or sensors, etc) are common, and don't seem to be well addressed by having all analysis start with the controller.

Page 185. From the analysis of hazardous system behaviors, component safety constraints

(also called “requirements” by Leveson) are inferred. Again, we are disturbed by the fact that the constraints on the controller appear to involve phenomena that it has only indirect access to (eg, “the door is open”). How exactly are these constraints obtained? Without a mechanical analysis, it seems to us very likely that some cases will be missed.

The constraints themselves appear to be mutually contradictory. For example, (2) says that a power off command must be provided within x milliseconds after the door is opened. So there may be a period during which the door is open and the power is on. But (1) says that the power must be off when the door is open. Similar issues seem to arise for power on, which, assuming it suffers similar delays (although this is not mentioned), would invalidate (3).

Page 187. Leveson now explains why it is not sufficient to hand the component safety constraints to the engineers and expect them to be fulfilled. As an example, she relates a problem with a system described by Kletz, in which a chemical reactor has two valves, one admitting catalyst to the reaction chamber and the other admitting water to a coolant chamber. The safety constraint on the software is that the water valve should be opened before the catalyst valve. The issue Leveson raises is that the command issued by the software to open the water valve may fail to have the desired effect, because the valve is stuck, or the actuator is broken, etc.

This indeed is an important concern, but from our perspective, it has nothing to do with the software. In our approach, in the same stage at which the software specification is proposed, a domain assumption is given for the valve, actuators, etc. This draws direct attention to the possibility of the valve failing. Moreover, the use of distinct phenomena for sending the commands to the valve and the valve actually opening or closing, and the demand that they be designated carefully, further focuses the analyst’s attention on the potential gap between what the software sees and what happens in the reaction chamber.

It seems to us that there is a lack of modularity in Leveson’s approach; the software safety constraint is evidently about not only the software, but also the valves, actuators, etc. In our approach, each domain is assigned properties that can be evaluated independently with respect to knowledge of that domain; that the components work together is factored out into a separate argument. In Leveson’s approach, these two aspects (component correctness and component interaction to satisfy system-level properties) do not seem to be clearly separated.

Page 188. Process models are given for the power controller and human operator.

It is notable that no models are given for the sensors and actuators, or for the power source itself. This seems to arise from the emphasis placed in Leveson’s approach on the control software; in our approach, the control software and its computer are treated at the same level as the sensors and actuators (and communication channels if used, and so on).

There doesn’t appear to be a consideration in Leveson’s analysis of initial conditions. As we noted in our analysis, we believe this is a fundamental concern that is often a source of problems that go unnoticed. Many failures seem to occur when systems are being restarted or reconfigured.

Page 189. A diagram is shown for exploring anomalous behaviors of components that might lead to violation of component safety constraints.

Note that the diagram includes both product issues (actuator delayed) and managerial process issues (flaws in creation of of control algorithm). On the one hand, we agree that it is important to include process issues. On the other hand, we have taken the position to date that these are sufficiently important and different that they should be analyzed separately. We would want to consider, in addition to the concerns that Leveson raises here, the possibility that the correctness arguments for components are not consistent with the deployed components because of a failure of version control, but there doesn't seem to be room for such a consideration here. There may well be interactions between process and product issues, but that doesn't mean it isn't better first to analyze them in isolation, and only then consider their interactions.

The idea of following the control actions around the system diagram is very appealing. We wonder, however, how systematically it can be done, and whether it is as effective as performing a mechanical analysis combining models of all components. It is certainly likely to be a less expensive analysis to perform. As a brainstorming exercise, it seems to be a very good idea, but we don't see how it can be made rigorous to ensure that important cases are not missed.

Note that in this diagram new phenomena seem to be appearing that suggest the presence of a communications channel (which is then made explicit on page 190) that introduces delays. Issues of this sort would have been identified earlier in our approach, since channels would have to be represented explicitly in the context diagram.

On page 191, Leveson raises for the first time the question of there being two operators. We believe this concern belongs at the start with the designation of phenomena. In our approach, the analyst is pushed towards facing the issue because in designating the events open, close and touch, the analyst must decide whether the operator performing the event should be identified as an argument of the event, or whether (as in this case) the events should all be performed anonymously, allowing the possibility that a different operator performs each event.

Summary

The main differences between the approaches seem to be:

1. *Explicitness.* In our approach, we attempt to make every assumption explicit. In Leveson's approach, assumptions are often implicit: for example, whether sensors are faithful, how many operators there are, whether operator can enter chamber and close door behind, etc. In addition, every step of our logical argument is made explicit -- and if a step is omitted, the mechanical analysis will likely catch it.

2. *Phenomena.* In our approach, early identification and designation of phenomena are central.

In Leveson's approach, the set of relevant phenomena seem to emerge during analysis. For example, the question of whether a communications channel is reliable (or even whether it is present) is addressed at the start; in Leveson's approach, this consideration seems to emerge during a later step when one considers how safety actions might fail to occur as desired. We think that identifying phenomena upfront is both easier and less error-prone than discovering them on-the-fly. As Michael Jackson noted (in a post to the safety critical group), there is always a risk that phenomena that have not been considered will turn out to be relevant. Being *less* explicit about phenomena does not seem to be a more likely way to find them.

3. *Requirements vs. specifications.* In our approach, there is a rigid distinction made between requirements (which constrain the phenomena in the world that are ultimately of interest) to specifications (which constrain the behavior of machines, and are described in terms of the phenomena at their interface alone). Leveson appears not to have this distinction, and the safety constraints for components are not restricted to talking about the phenomena at the interfaces do those components.

From our perspective, this muddies the water with respect to the question of who is responsible for what. As Butler Lampson noted in his famous *Hints* paper:

"There probably isn't a best way to build the system, or even any major part of it; much more important is to avoid choosing a terrible way, and to have a clear division of responsibilities among the parts."

4. *Articulating and reasoning about the hazard.* In our approach, since the critical requirement (which Leveson calls the "hazard") is ultimately what matters, we believe it should be articulated as precisely as possible, using designated phenomena, and that the dependability argument should demonstrate that the requirement itself is a consequence of the component and domain properties. Although Leveson states the hazard, the analysis is formulated in terms of the "system safety constraint", and there is no explicit consideration of the hazard. In other words, there is a dangerous gap in Leveson's approach between the hazard and the system safety constraint, which is not addressed by rigorous analysis. For example, while in our approach we formalize the notion of touching the power source and include an explicit model of how this event might be interleaved with open and close events, in Leveson's analysis there is no explicit representation of this event, and it is not included in the analysis.

5. *Top-down analysis.* In Leveson's approach, the analysis begins with the potentially hazardous actions, and then explores how they might occur. Our approach is similarly top down in the sense that the machine specifications and domain assumptions are selected in the hope of satisfying the requirement, but our analysis itself is not top-down. Instead, we advocate constructing an argument that considers all possible executions and shows that the requirements follows from the specifications and assumptions. In critical systems, we believe that this argument should be checked mechanically, to increase confidence in its soundness, and that the cost of such an argument should not be prohibitive since the domain assumptions and specifications are partial.