

Introducing Semi-Formal Specifications into a Distributed Development Process

[Extended Abstract]

Hardi Hungar and Michael Meyer zu Hörste
Institute of Transportation Systems
German Aerospace Center
Brunswick, Germany
{hardi.hungar,michael.meyerzuhoerste}@dlr.de

ABSTRACT

In Europe, the development of safety-critical rail IT is regulated by the Common Safety Method (CSM 352) and CENELEC standards EN 50126 to EN 50129. The CSM requires the railway undertaking to unambiguously specify requirements on equipment. The infrastructure branch of German Railways (DB Netz) has chosen to base the specifications of interlockings and related equipment like field element controllers on SysML statecharts. Although these specifications will be more consistent and precise than the text-based ones in use before, this approach faces opposition from manufacturers who find it difficult to justify the correctness of their implementation.

This paper reports on the approach, its motivation and its practical difficulties, and it endeavours to identify its merits and weaknesses.

Keywords

formalisation, development process, standards

1. BACKGROUND

To have an unambiguous, consistent specification at a point in a development process where the manufacturer takes over from the purchasing railway seems like a natural, beneficial thing. The *Common Safety Methods* CSM 352/2009 and its revision CSM 452/2013 include this as a requirement on a sound development for safety-critical rail systems.

The legal frame for the implementation is given by a number of standards: CSM and CENELEC standards are focussed on safety. In parallel the so-called TSI (technical specification for interoperability) are giving the frame for the functional harmonisation over Europe. Further—typically national—standards are focussing on the serviceability and the process of taking into operation. All of them have in common that they need to be precise and unambiguous.

2. SPECIFICATION STYLES: OPERATIONAL VERSUS PREDICATIVE

SysML statecharts [3] provide a dialect of statecharts [4] for specifying behavior on system level. There is not much particular about this dialect to make it more suitable for this task than any other statecharts dialect: In general, SysML statecharts may be characterized as a way to define parallel processes with a hierarchical structure to name and organize control states. Communication may be done through parameterized events and shared variables. They have a sophisticated runtime execution whose details are to be fixed by each particular implementation.

The resulting specifications are operational in nature. In other words, the specifications are given in a form of programs. A statechart system, in reaction to external input—usually events—changes internal states and computes outputs in a rather deterministic, step-by-step fashion. The properties of the specified system result from the set of possible behaviors. And a complex specification is as hard to analyse for properties as a program.

In contrast to that, a predicative specification would define properties of the system. This would enable to say, for instance, that a particular reaction to some high-priority input event would have to occur within a fixed time frame. On the other hand, it would be difficult to see that a collection of properties is free of contradictions. And to define some behavioral sequence would be very cumbersome.

These observations hold as well for informal specification styles as for formal ones, and they are well known for decades. The decision for one of these two—or a mixture of both—has consequences on how an implementation is proven correct: For a (pure) declarative specification, one has to establish properties of software or software/hardware combinations. If the specification is operational, the task takes on the form of establishing a *refinement proof*. We will discuss these questions for the present approach of the German Railways.

3. THE MANUFACTURER POINT OF VIEW

Manufacturers of railway systems obviously have established ways of dealing with traditional rail system specifications in their respective development process. These specifications usually are a mixture of properties to be achieved, behavior (sequences) of typical and failure-related nature, and techni-

cal details. Those requirements can be, more or less, decomposed into single items permitting a tracing throughout the development process. Since they are not formalised, there is no need to fully rule out inconsistencies on specification level, or to make the specifications unambiguous and complete. Anything left open or unclear is solved by the developers. Detailed knowledge and long experience enable them to deliver functional and safe equipment. Verification and validation rely heavily on tests, complemented by arguments and analyses.

With a statechart specification, tracing becomes much more difficult, as there are no single requirements to which tests may refer. So the usual design verification, which is documented by relating evidence (e.g., test results) to properties to be checked, does not work anymore. Instead, it must be proven that the input/output behaviors of the implementation and the statechart specification are the same. Where it might not even be possible to really achieve “the same” behavior in an implementation, for instance, if timing is an issue. And it will be very difficult to argue that some particular deviations which have been found are not critical, because properties are specified indirectly and are not qualified or explained as it is (sometimes) done in more traditional specification styles.

These objections, though partly rooted in the difficulty to adopt a new way of development, raise important methodological questions. These will be discussed in more detail in the following section.

4. ESTABLISHING REFINEMENT RELATIONS

4.1 Refinement in General

The task of proving the correctness of an implementation is an instance of the well-known problem of establishing a refinement relation. Two systems $S_1 = (I, O, V_1)$ and $S_2 = (I, O, V_2)$ having the same external interface of input and output variables I and O (or channels, or any other name) but different internals V_1 and V_2 are behaviorally equivalent if

$$\llbracket S_1 \rrbracket_{I,O} = \llbracket S_2 \rrbracket_{I,O} , \quad (1)$$

i.e., if their behaviors agree on the external variables. Refinement means behavioral inclusion:

$$\llbracket S_2 \rrbracket_{I,O} \subseteq \llbracket S_1 \rrbracket_{I,O} . \quad (2)$$

in the formulas above, $\llbracket S \rrbracket$ may denote a set of traces as the semantics of a system S . Given a domain for time $\mathcal{T}$ and one for variable values $\mathcal{V}$, the set of traces $\mathcal{S}$ are the mappings

$$[\mathcal{T} \rightarrow [\mathcal{X} \rightarrow \mathcal{V}]]$$

which assign to each point in time a variable valuation. Equivalently, we can view each variable semantically as a sequence of values, arriving at

$$[\mathcal{X} \rightarrow [\mathcal{T} \rightarrow \mathcal{V}]]$$

as semantical domain, and treat the semantics of a system like a logical formula. Then, refinement is expressed by the logical formula

$$\exists V_2 . \llbracket S_2 \rrbracket \Rightarrow \exists V_1 . \llbracket S_1 \rrbracket . \quad (3)$$

This is essentially the form in which Lamport [5] studied refinement in his “temporal logic of actions”.

To actually prove that S_2 is a refinement of S_1 , the most common practical way is to define a mapping from the internal variables of S_2 to those of S_1 (or, more general, a relation between them) and show that this mapping induces a simulation. It may be said that this “practical way” is already rather difficult. And there are even more general forms of refinement, cf. [1], which are not easier to prove.

For a first discussion of the problem at hand, functional system specifications in the form of SysML statecharts, the notion expressed in (3) can be taken as a preliminary definition. As said above, the manufacturers consider it a very difficult task to establish this relation.

4.2 Refinement in Practice: Model-Based Development

Having to show the correctness of some claimed refinement step is not a new concept in safety-critical system development: Each step towards an implementation has to be verified. For instance, when the architecture of a software is defined and the specifications of its components are designed, it has to be shown that the software requirements are correctly refined. It is the form of the requirements which is the cause for the difficulties, here.

Though model-based techniques are not adopted universally in the safety-critical domain, there are numerous examples of their application. It makes a big difference, however, to introduce model-based techniques within one organisation, or to install them at an organisational boundary. A rather widespread mode of how to develop within a model-based way is to iteratively and incrementally refine some abstract design model. The end result of this continuous refinement is the implementation. From the very concept of that idea, high-level states are *refined* into lower-level ones. This need not be a strict refinement in the mathematical sense above. But essentially, the development produces a chain of refinement maps from the abstract to the concrete layer.

Comparing this with a situation where some manufacturer has a line of products verified (in the practical meaning of that word) against standard, text-based specifications, and the next version shall be proven consistent with a statechart specification. Usually, there will not be any “simple” refinement map, as this would only be available if the product architecture matches the one of the model. More often than not, for any system of nontrivial complexity, the implementation mechanics will differ considerably from that of the specification model, and the existence of a useable refinement map is highly improbable.

4.3 Refinement in Practice: Verification by Testing

Testing is the classical approach to verify properties of a system. This could be done either in the field or in lab. The different aspects are usually addressed in specifically targeted tests:

Safety: Tests focussed on extreme or erroneous behaviour

Conformity: Tests focussed on regular and typical functional behavior

Serviceability: Regular operational cases as well as operationally relevant disturbed cases.

These categories, with which all actors in the field are well acquainted, apply to *complete* systems. They describe in which respect a system is validated once it is developed. A safety validation step is required for any safety-critical system by the standards. During development, white box tests for unit verification and gray-box test for integration checks are applied. In the common depiction of standard development processes as a V, these tests start at the bottom (unit tests) and are applied throughout the phases on the right.

Here, we use the terms “verification” and “validation” in their technical meaning as it is defined in the CENELEC standards: *Verification* serves to establish that the output of one development step is an implementation of its input requirement, i.e., the output of the previous step. For instance, as a first step in a development, one may define the system architecture and apportion the requirements to the system components. Then, it must be *verified* that the combination of the components according to the architecture will satisfy all input requirements. *Validation* checks the full system against user requirements. More general, validation is also used to denote checking whether some design artifact conforms to a specification given several steps earlier. One may, for instance, validate the software of the safety kernel after integrating its component modules against safety properties.

In model-based development as described above, testing begins already on the left side of the V. One may perform verification as well as validation already in earlier phases. This becomes possible if intermediate development artifacts take the form of executable models.

In the strictly regulated development of safety-critical rail systems, for each step a thorough verification method has to be chosen to stand a chance of getting the result certified for field use. We would consider systematic testing as a promising approach.

We assume that the specification is an executable SysML specification composed of a few block diagrams whose behavior in turn is given by statecharts. Some of the specifications already satisfy this assumption, the others are intended to be advanced to this state. Then, one may see these specifications as extended finite state machines (E-FSMs).

A method for checking behavioral conformance¹ of some system with an FSM (propositional, not “extended”) has been described by Chow in [2]. Essentially, one checks that each transition in the FSM is matched by an action the implementation. Lifting this idea to extended FSMs, one would not only check each transition, but also take the set of variable valuations into account, for instance, by requiring modified condition/decision coverage (MC/DC). This is what advanced commercial test generation tools promise to auto-

¹In this case, “behavioral conformance” is to be interpreted as conformance with the full behavior spectrum of the requirement model—for implementing a model, *all* of the model’s behavior is to be considered “regular”.

mate to a large extent. Rhapsody Automatic Test Generation (ATG) and RT-Tester are just two examples of such tools.

A test suite which covers the requirement specification thoroughly can then be used to verify that some other executable design model, or a program, conforms behaviorally to the requirement SysML-model. Even if the architecture of the implementation differs considerably from that of the requirement—what would complicate a refinement proof, even an informal one—, a successful test would provide useful evidence for arguing the correctness of the implementation.

5. CONCLUSION

The introduction of semi-formal specifications of operational nature in the form of SysML statecharts confronts manufacturers with a serious verification problem. Constructing a test suite to cover the specification provides a way to produce evidence for a correctness argumentation. There are caveats, of course.

First, the implementing party would be forced to continue the development with executable artifacts (models) on which those test suites can be run. Second, that a successful test of behavioral conformance with the requirement model will be accepted as sole evidence is by no means clear. Perhaps, or even probably, additional verification activities like early validation of initial operational requirements (serviceability) might be required. And third, discrepancies in the behavior will occur nearly inevitably. Few implementation steps which introduce more detail can be done while retaining full equivalence. And even if a manufacturer just constructs a requirement model fitting his intended architecture, deviations in timing or sequence of reactions will be hard to avoid. And it is difficult to argue that some deviation is uncritical, if the specification is not given in the form of properties.

Summarizing, the formalisation introduced into the requirement phase poses indeed a severe problem for the manufacturers. And though there seems to be an approach based on systematic testing which could contribute substantially to a practical solution, one might suppose that the change in requirement style will have considerable impact on the development process as a whole.

6. REFERENCES

- [1] Manfred Broy. Compositional refinement of interactive systems. *J. ACM*, 44(6):850–891, 1997.
- [2] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. on Software Eng.*, 4:178–187, 1978.
- [3] Object Management Group. OMG System Modeling Language. electronic, 2012. <http://www.omg.sysml.org>.
- [4] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comp. Programming*, 8:231–274, 1987.
- [5] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16:872–923, 1991.