IMEKO TC6 International Conference on Metrology and Digital Transformation
September 19 – September 21, 2022, Berlin, Germany

# SPECIFICATION FOR AUDIT TRAIL IN OIML D31: TOWARD RUNTIME VERIFICATION

*Hiroshi Watanabe* [a,*]

[a]National Metrology Institute of Japan (NMIJ), National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba Central 3, Tsukuba 305-8563, Japan
[*]Corresponding author. E-mail address: hiroshi-watanabe@aist.go.jp

***Abstract*** – In general, manually checking a system log does not scale up, if the log becomes large. The problem also arises the case with checking an audit trail in verification of a measuring instrument under legal control. Software support for the check is critical. In this study, we address an approach to use Runtime Verification techniques for checking an audit trail. As an initial attempt to advance the approach, we consider the formal specifications of an audit trail for the check. Analysing the requirements in guidance OIML D31, we obtained abstract formal specifications for an audit trail. As an extension of the approach, a trend toward digital transformation for audit trails can be envisaged.

***Keywords:*** legally relevant measuring instrument, OIML D31, audit trail, verification, traced update, Runtime Verification

## 1. INTRODUCTION

Software-controlled measuring instruments under legal control have audit trails, which are used as both evidence of and a protection measure against intervention. Checking an audit trail is concretely required as a test item in legal[1] verification in guidance OIML D31 [1, 2]. The check will soon be conducted under the existing legal verification. The check itself is designed to be run manually; however, manual checking does not scale up. Consider, for example, a legal verification personnel (who might not be an information technology expert), checking an audit trail line-by-line at a simple instrument display or on printouts at a site. Software support for checking audit trails is critical.

Our primary question is whether we can use Runtime Verification [3, 4] for the check. Runtime Verification (RV) is a verification technique that symbolically checks whether a trace (i.e., finite sequence of recorded events) of a system under scrutiny satisfies (or violates) a given formal specification for the trace. RV also includes a technique or tool that generates software called a monitor that can be built into the system and execute the checking. RV has emerged from symbolic model checking [5] in the field of formal methods, and has advanced over the past 20 years [6]. For describing the formal specification for the trace in RV, linear temporal logic (LTL) [7] is most commonly used.

As an initial attempt toward using RV, we analyse the requirements of audit trails in the current 1CD [2] of OIML D31[1] and consider formal specifications for these requirements in LTL. As a result of our analysis, we obtain abstract formal specifications with respect to test items for legal verification in section 4.1. Moreover, we obtain the specifications for other relevant requirements, mostly Traced update, in section 4.2. We discuss our contribution and the direction of digital transformation for audit trails in section 5. In the present study, we do not yet delve into the RV itself.

[1]For denoting a verification of a measuring instrument in legal metrology, we prefix "legal" to the verification in this paper.

## 2. OIML D31 AND AUDIT TRAILS

OIML D31 [1] lists requirements for software-controlled measuring instruments with respect to the following topics: Software for the measuring instruments; Type evaluation; and legal Verification. D31 is a guidance document of OIML for implementing international recommendations for individual measuring instruments. The latest version [1] was published in 2019. The revision process is ongoing, and we use the current 1CD [2] for the analysis in the present study.

An audit trail is "*continuous data containing a time stamped information record of events* [2, 3.2.1]". The requirements in 1CD are summarized in the following sections 2.1, 2.2, and 2.3.

### *2.1. Events to be recorded in an audit trail*

First, the following events are required to be recorded in an audit trail:

- changes in the value of the legally relevant parameters [2, 3.2.19 and 6.2.3.6];
- modifications or updates of the legally relevant software [2, 3.2.19], especially the record of Traced update including the information of success/failure [2, 6.3.8.4.1 and 6.3.8.4.8]; and
- other activities that are legally relevant and that might influence the metrological data/or characteristics, as follows: all changes for specified parameters of dynamic modules [2, 6.2.3]; exchange of components [2, 6.3.2.1.3 Examples 2)]; modification of access permissions of legally relevant files [2, 6.3.6.5.4 Examples 1)]; and changes to the operating system configurations [2, 6.3.6.7.2 Example].

### 2.2. *Test items in legal verification*

There are three (actually two) test items with respect to an audit trail, as follows:

1. The parameters requiring integrity should be checked by the entries of audit trails as *check the audit trail for events concerning parameters* [2, 8.2.3.2].

2. Actually, the requirement is not to check the audit trail but to *check whether all settable parameters are within the allowed range* [2, 8.2.3.1].

3. *The entries of the audit trail for traced updates* should be checked in the Identity of the software [2, 8.2.4].

No further information about concrete methods for checking audit trails is available in D31 except a clause related to documentation, which states that *securing means as well as means to provide evidence of an intervention and the method to check them (e.g. hardware seals, event counters, audit trails)* shall be included in the certification [2, 7.2.2]. This statement implies that type-wise concrete methods are left to be developed and provided by manufacturers.

### 2.3. *Other relevant requirements: Traced update*

In addition to Verified update, Traced update is the other software update procedure without subsequent legal verification after the update. Its requirements potential for check by audit trail are summarized as follows:

1. The traced update procedure is specified in [2, 6.3.8.4] with a flowchart [2, Figure.1].

2. Depending on national legislation, user or owner consent is required before proceeding with an update process [2, 6.3.8.4.2].

3. *If some of the securing or protection measures of the instrument are turned off to enable updating, they shall be turned on again immediately after update, independent of the result of the update process.* [2, 6.3.8.4.3].

4. Switching to an inoperable mode after failure of either the integrity test or the authenticity test is required [2, 6.3.8.4.7].

In the present study, we do not address the requirements related to the following topics in revision [2]: remote verification and dynamic modules of legally relevant software.

## 3. LINEAR TEMPORAL LOGIC (LTL)

The most common specification language used for specifying traces in RV is LTL, which was proposed by Pnueli [7]. Here, we briefly introduce a fragment of LTL for presenting our results in section 4. More detailed descriptions are available in the literature, such as [3, 4, 5].

LTL[2] is an extension of propositional logic with temporal connectives: unaries $\mathbf{G}$, $\mathbf{F}$, and a binary $\mathbf{U}$ . Here, $\mathbf{G}$ stands for "**G**lobally",$\mathbf{F}$ for "**F**uture", and $\mathbf{U}$ for "**U**ntil".

We here introduce some terminology related to traces—finite sequences of events—before we describe the semantics of LTL. The successor of a trace is the rest of it except the first event. For any trace $t$, the $t$ itself and all successors up to the final successor consisting of a single event are called descendants of $t$.

In RV, formulas of LTL are interpreted over traces to have a value of true or false. We assume that atomic propositions represent properties of events. Let $t$ be a trace. An atomic proposition $p$ is true at $t$ if the first event of $t$ has the property $p$. For connectives for propositional logic, the usual interpretation can be extended naturally over traces. Finally, the interpretation for temporal connectives is given for arbitrary formulas $P$ and $Q$ as follows:

- $\mathbf{F}P$ is true at $t$ iff $P$ is true at some descendant of $t$;
- $\mathbf{G}P$ is true at $t$ iff $P$ is true at all descendants of $t$; and
- $P\ \mathbf{U}\ Q$ is true at $t$ iff $Q$ is true at some descendant and $P$ is true at all descendants before the descendant.

We present some typical examples of LTL formulas as follows:

| | |
|---|---|
| *Globally, $P$ is false* | $\mathbf{G}\neg P$ |
| *Globally, $Q$ responds to $P$* | $\mathbf{G}(P \rightarrow \mathbf{F}Q)$ |
| *$P$ is always false between $Q$ and $R$* | $\mathbf{G}((Q \wedge \mathbf{F}R) \rightarrow ((\neg P)\ \mathbf{U}\ R))$ |
| *$P$ becomes true between $Q$ and $R$* | $\mathbf{G}((Q \wedge \mathbf{F}R) \rightarrow ((\neg R)\ \mathbf{U}\ P))$ |

The first type of formula is called a safety property; it ensures something undesirable (here, the property $P$) never occurs. The second type of formula is called a liveness property; it ensures that something desirable (here, the property $Q$) will occur eventually. For other examples of LTL formulas, we refer the reader to the well-known specification patterns of Dwyer et al. [8].

## 4. FORMAL SPECIFICATION FOR AN AUDIT TRAIL

Analysing the requirements in OIML D31, we obtained abstract formal specifications for an audit trail, which we present in the following sections 4.1 and 4.2. We refer to the specifications as "abstract" because, to represent the requirements as formulas of LTL, we introduce abstract properties, such as the terms $changeParameterRequiringIntegrity$ and $changeParameterOutOfRange$ in Tables 1 and 2. Each of these properties corresponds to an event or a property of an event either stated in or inferred from the requirements. In the following sections, we omit detailed explanations for individual abstract properties because of space constraints.

[2]We omit the next time operator $\mathbf{X}$ for simplicity of our presentation of this work.

Table 1: Abstract formal specifications for test items in section 2.2

8.2.3.2 Integrity. Any change of parameter requiring integrity never occurs.
$\mathbf{G}\neg changeParameterRequiringIntegrity$ (1)

related to 8.2.3.1 Correctness. Any change of parameter out of allowed range never occurs.
$\mathbf{G}\neg changeParameterOutOfRange$ (2)

8.2.4 Identity of the software. Any traced update log with entry of illegal software identification never occurs.
$\mathbf{G}\neg tracedUpdateLogIllegalSoftwareIdentification$ (3)

example related. While maintenance mode, no measurement succeeds to produce a measurement result.
$\mathbf{G}(startMaintenanceMode \wedge \mathbf{F}startNormalMode \rightarrow \neg successMeasurement\ \mathbf{U}\ startNormalMode)$ (4)

example related. While normal (operable) mode, any unauthorised change of parameter or software never occurs.
$$\mathbf{G}(startNormalMode \wedge \mathbf{F}(startMaintenanceMode \vee startUpdateProcess) \rightarrow \neg changeParameterorSoftwareUnauthorized\ \mathbf{U}\ (startMaintenanceMode \vee startUpdateProcess))$$ (5)

Table 2: Abstract formal specifications for Traced update in section 2.3 and others.

6.3.8.4 and Figure 1. Obeying the procedure of Traced update in the flowchart [1, 2, Figure 1].
$\mathbf{G}(requestUpdate \rightarrow \mathbf{F}(ownerConsent \vee ownerNotConsent))$,
$\mathbf{G}(ownerNotConsent \rightarrow \mathbf{F}returnToNormalMode)$,
$\mathbf{G}(ownerConsent \rightarrow \mathbf{F}fileLoaded)$,
$\mathbf{G}(fileLoaded \rightarrow \mathbf{F}(validIntegrity \vee invalidIntegrity))$,
$\mathbf{G}(validIntegrity \rightarrow \mathbf{F}(validAuthenticity \vee invalidAuthenticity))$,
$\mathbf{G}(validAuthenticity \rightarrow \mathbf{F}installActivate)$,
$\mathbf{G}((invalidIntegrity \vee invalidAuthenticity) \rightarrow \mathbf{F}(discardFilesKeepOld \vee switchInoperableMode))$,
$\mathbf{G}((installActivate \vee discardFilesKeepOld \vee switchInoperableMode) \rightarrow \mathbf{F}logTracedUpdate)$,
$\mathbf{G}(logTracedUpdate \rightarrow \mathbf{F}reboot)$ (6)

6.3.8.4.2. The user or owner expresses their consent before the measuring instrument starts update procedure.
$\mathbf{G}(requestUpdate \wedge (\mathbf{F}startUpdateProcess) \rightarrow \neg startUpdateProcess\ \mathbf{U}\ ownerConsent)$ (7)

6.3.8.4.3. The protection measures are turned on again before an event representing "immediately after update".
$$\mathbf{F}turnOffProtectionMeasure \rightarrow \mathbf{G}((installActivate \vee discardFilesKeepOld \vee switchInoperableMode) \rightarrow (\neg immediatelyAfterUpdate\ \mathbf{U}\ turnOnProtectionMeasure))$$ (8)

related to 6.8.3.4.3. While the protection measures are turned off, any unauthorised change of parameter or software never occurs. Here, the unauthorised change excludes the one caused by the update procedure itself.
$$\mathbf{G}(turnOffProtectionMeasure \rightarrow (\neg changeParameterorSoftwareUnauthorizedExceptUpdate\ \mathbf{U}\ turnOnProtectionMeasure))$$ (9)

6.3.8.4.7. Failing either test, without switching to inoperable mode, the measuring instrument discards the update files.
$$\mathbf{G}(((invalidIntegrity \vee invalidAuthenticity) \wedge (\neg switchInoperableMode\ \mathbf{U}\ endTracedUpdate)) \rightarrow (\neg endTracedUpdate\ \mathbf{U}\ discardFilesKeepOld))$$ (10)

6.3.8.4.7. While inoperable mode, the measuring functions are inhibited.
$$\mathbf{G}(switchInoperableMode \rightarrow (measuringFunctionInhibited\ \mathbf{U}\ (startNormalMode \vee verifiedSubsequently \vee (\mathbf{G}measuringFunctionInhibited))))$$ (11)

6.3.2.1.3 Example 2). Any unauthorised exchange of components never occurs.
$\mathbf{G}\neg exchangeComponentUnauthorized$ (12)

6.3.6.5.4 Example 1). Any unauthorised change of access permission never occurs.
$\mathbf{G}\neg changePermissionUnauthorized$ (13)

### 4.1. *Specifications for test items in legal Verification*

For test items described in section 2.2, we obtained a list of abstract specifications, represented as formulas of the safety property, (1), (2)[3] and (3) in Table 1. As related to these specifications, we obtained additional examples of abstract specifications (4) and (5).

### 4.2. *Specifications with respect to Traced update and others*

For the Traced update described in section 2.3, we obtained a list of abstract specifications (6), (7), (8), (10) and (11) in Table 2. As related to (8), we obtained an additional example of specification (9) that any unauthorised change of a parameter or software never occurs while protection measures are turned off. We remark that the formulas of (6) are directly extracted from the flowchart [2, Figure 1] describing the Traced update procedure.

With respect to both exchange of components [2, 6.3.2.1.3 Example 2)] and modification of access permissions of legally relevant files [2, 6.3.6.5.4 Example 1)] described in section 2.1, we obtained abstract specifications (12) and (13), respectively.

## 5. DISCUSSION

Our contribution is summarized as follows:

- The obtained abstract formal specifications in section 4 will provide detailed insights into the design required for both RV and audit trails in measuring instruments. These insights will lead to the concrete implementation of the approach of using RV for checking audit trails.
- The most important outcome is that our abstract formal specifications will be available in the practical check by instantiating the abstract properties in the specifications.
- Some readers might assume that the general query in database management systems is sufficient if only the test items described in section 2.2, which are represented as safety-property formulas (1), (2) and (3) in Table 1, are checked. Although we agree with the observation, we do not necessarily need to restrict ourselves only to the test items. Our formulas suggest that the check can be extended seamlessly toward other relevant requirements (e.g., formulas in Table 2) without changing the framework.

For future work, we envisage the following directions of digital transformation for audit trails:

- Use of RV for checking audit trails. As a part of the documentation for type examination, manufacturers instantiate our abstract specification for an audit trail of their type and submit it for approval. The specification is examined by relevant authorities under the process of type evaluation. The approved formal specification is used for legal verification for a specimen of the type. For this purpose, future research opportunities include developing, for example, a standard format and evaluation methods for audit trails.
- Use of RV for generating legally relevant monitors for audit trails. As a potential next stage beyond mere check, the legally relevant monitors will be built into measuring instruments enabling them to work as runtime checking facilities.

## 6. CONCLUSIONS

Analysing the requirements in OIML D31, we obtained abstract formal specifications for audit trails; the specifications were obtained as LTL formulas that will be used in the RV framework. Our obtained specifications suggest explicitly the remaining work for designing and implementing the approach using RV for checking audit trails. Moreover, a progression toward digital transformation for audit trails can be envisaged.

## REFERENCES


[1] OIML D31:2019, General requirements for software controlled measuring instruments. https://www.oiml.org/en/publications/documents/en/files/pdf_d/d031-consolidated-e19.pdf (accessed 25 March 2022).

[2] OIML TC 5/SC 2/p 4, Revision of OIML D31: General requirements for software controlled measuring instruments, 1CD, 2021. https://www.oiml.org/en/tc-sc-pg/committee-drafts/files/tc5-sc2-p4-1cd.zip (accessed 25 March 2022).

[3] E. Bartocci, Y. Falcone, A. Francalanza and G. Reger, Introduction to Runtime Verification, in: E. Bartocci and Y. Falcone (Eds.), Lectures on Runtime Verification, LNCS, 10457, pp. 1-33, 2018.

[4] M. Leucker and C. Schallhart, A brief account of runtime verification, J. Logic Algebraic Program., vol. 78, no. 5, pp 293-303, 2009.

[5] E. M. Clarke, O. Grumberg, D. A. Peled and H. Veith, *Model Checking, Second Edition*, MIT Press, 2018.

[6] Runtime Verification, https://runtime-verification.github.io/ (accessed 25 March 2022).

[7] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 46-57, 1977.

[8] M. B. Dwyer, G. S. Avrunin and J. C. Corbett, Property specification patterns for finite-state verification[4], in: Proceedings of the second workshop on Formal methods in software practice (FMSP '98), ACM, New York, pp. 7-15, 1998.


[3] Although (2) is not actually required in test items, as we pointed out in section 2.2.2, we include it as an item.

[4] see also A Specification Pattern System, 1998, https://people.cs.ksu.edu/~dwyer/spec-patterns.ORIGINAL