Published on *OASIS* (https://www.oasis-open.org)

Interoperability Guidelines

Description:
Describing best practices for writing specifications, so that the risk of having interoperability (or portability) failures between implementations is reduced

This document provides guidelines about best practices in writing specifications, so that the risk of having interoperability (or portability) failures between implementations is reduced. The target audience is primarily specification writers and TC members.

Status: TAB-approved deliverable.

Editor(s): Jacques Durand

Table of Contents

1 Interoperability Defined

Interoperability is understood in different ways depending on the specification under consideration:

- If the specification is about a communication protocol (e.g. a transfer protocol, an interface) and about the behavior of processors of this protocol, then interoperability is understood as the ability of two implementations of this specification ? i.e. processors of this protocol - to communicate properly. In the case of an interface, ability of a user entity to communicate with an implementation (or processor) of the interface.
- If the specification is of an artifact (e.g. a business document, a process definition, a policy contract), then interoperability is understood as the ability to process this artifact with consistent results, using different platforms or processors. In such a case, interoperability is often described as*portability* from the artifact perspective (the artifact is portable across platforms), while the platforms or processors are qualified as interoperable.

Often, specifications mix both aspects. Interoperability is generally described as*portability* when an*entity* - an artifact, a software component - is expected to be used over a*platform* - an API, a processor, a framework. The standardization of this platform (and of some aspect of the entity) will help the portability of such an entity over different instances of the platform.

2 Interoperability Guidelines: Useful Where and When

These guidelines concern the writing of specifications where there is a clear expectation that these specifications are precise enough to ensure that their implementations are interoperable or very close to be. In other words, these guidelines apply when interoperability is primarily a matter of correct or consensual interpretation of the specification.

This is not always the case:

- Standardization may concern existing technologies, where there are pre-existing implementations that are seeking to converge or interoperate. Multiple vendors will attempt to converge based on the same basic

technology. This is as much a political process as a technical one, based on negotiations where convergence may reach some limits due to cost or difficulty to depart too much from legacy implementations.
- Standardization may concern a brand new technology, even speculatively, where no implementations exist yet, or the realization of which is still conditional to unknown factors. In this situation it may not be possible to know how sufficient is the specification for interoperability of its future implementations.
- Standardization of a technology where there is only initially a single implementation. This is the vendor-submission case, quite common. Depending on the composition of the TC, the resulting specification might be tightly bound to the needs of that one vendor, or made broader (more extensible) in order to encourage other vendor implementations. here again the specification may be relaxed to allow for variations in implementations, the convergence of which (to some acceptable interoperability level) is the object of an ulterior process - e.g. profiling.

3 Whose Responsibility?

A large part of the confusion about Interoperability can be summarized by the question:

- Which aspects of interoperability fall under (a) specification writers responsibility, and (b) implementation developers responsibility ?

Too often interoperability problems arise when each party (the specification authors, the implementation developers) is over-reliant on the other party to ensure interoperability.

This little guide is intended to point at the most common interoperability traps that specification writers could avoid, or could help implementers avoid. In other words, although the responsibility for these interoperability traps may not always be clearly attributed (in general, both parties share the blame in various proportions), there is always*something* that specification writers can do about them.

4 Ten Mistakes That Will Cause Interoperability Problems

4.1Mistake #1: Errors are for Wimps

Too often, the causes of errors that an implementation may experience are insufficiently identified and covered in a specification. Errors are also a communication item in addition to drawing a clear line between expected and unexpected behavior or feature. Yet they have low priority on the agenda of specification writers, when the schedule is tight. Proper definition, standardization and reporting of errors is an important aspect of

interoperability.

Best Practice:

- A specification should include an ?Errors? section that describes a list of errors. Each error type should be clearly identified, and an appropriate error message defined. For every specification requirement, it should be clear which error type applies in case of failure. Details about how the error is reported or transmitted may be left to subsequent implementations or profiles to decide. However, when several parties are involved, it is recommended to also specify:
- Expected behavior of a device / implementation when receiving an error notification from another device,
- Expected behavior of an implementation after generating such an error as result of processing a faulty artifact.

4.2Mistake #2: Remind me which Version of the Specification am I using right now?

Version and revision numbers of the implemented specification should be readily accessible, not just by other implementations but by end-users. Not doing so will invariably cause misunderstandings and interoperability issues over time. One may wonder: isn?t that mostly an implementation issue? Only partly so: while it is the responsibility of an implementation a specification to announce in some way which version(s) it supports (see the well known -v option in many command-line software tools), the specification should help make the version information that is associated with an artifact easy to access by an implementation. And when two implementations communicate, they also need a (standard) way of accessing version information about each other.

Best Practice:

- Whenever a specification concerns an artifact that is to be exchanged between devices or is portable across devices, the version information (including Revision #) should be associated in some way with this artifact, i.e. either appear in the artifact itself or be communicated on demand or prior to the exchange using other (specified) means. In XML artifacts, a different namespace and/or a schema may be associated with each version or revision. However, this technique may not always prove practical and could generate unexpected constraints. An explicit version/rev element is often the best way to convey version information in artifacts, as well as a distinct command or option in processors.

4.3Mistake #3: When I see ?MAY? or "SHOULD", I see ?optional? all the way

The optional character of a specified feature, when concerning an artifact that may be produced and consumed, is a common source of confusion and interoperability failure. An artifact MAY implement a feature. This clearly means that a compliant device producing such an artifact MAY omit this feature. However, any compliant device consuming such an artifact MUST be able to process this feature, should it be present (unless specified otherwise). This is the intention of the definition of the MAY / OPTIONAL keywords as stated in [RFC2119] (although this subtle interpretation is not explicit enough for SHOULD / RECOMMENDED):

- "...An implementation which does not include a particular option MUST be
  - prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. "

So implicitly, a MAY / SHOULD for an artifact becomes a MUST (be able to process) for its processors.

Best Practice:

- Add a section in your specification that could be titled: ?General Rules for Normative Interpretation?, and that clarifies how the usual ISO or RFC keywords that indicate optionality (OPTIONAL, SHOULD, RECOMMENDED, MAY?) should be interpreted from the viewpoint of a compliant artifact vs. of a processor of this artifact. It is also good practice to identify at the beginning of your specification, what are the main*conformance targets* that will be addressed. For example, a messaging protocol standard will make statements about (a) the message, (b) the sending software, (c) the receiving software. When stating a requirement - in particular an optional requirement - one must be explicit about which conformance target is concerned: e.g. is it for an artifact or for a processor(s) of this artifact?

4.4Mistake #4: I am expecting you to support this Optional Feature, will you ?

Often, specification writers use recommendations (SHOULD, RECOMMENDED) as a trade-off: on one hand they would like implementers to implement the recommended features as they know it would be best for interoperability or performance if everyone supports it. On the other hand making that feature mandatory may hurt a broader adoption of the specification. The end result is that some users will implement it and will expect others to do so, while others don?t.

Best Practice:

- *For specification writers*: This is where conformance clauses and conformance profiles matter. A conformance profile is typically defined by a conformance clause added to the specification. Such a clause defines precisely what it takes to conform to a particular ?profile? of the specification. The clause identifies a subset of the normative requirements that must be complied with - e.g. depending on the nature of the conformance target concerned by this profile: processor, document... or depending of some level of usage. In particular a conformance profile will take a stance about optional features: some will explicitly be left out, others will become mandatory for this conformance profile. The main body of the specification keep all these options open and make it possible for other conformance profiles to take a different stance on these.

For users and developers: The advantage of such profiles (and conformance clauses) is that they can be defined separately from a specification ? and afterward, as they often require some knowledge about usage that may not be known from specification writers. The conformance profile may also focus on the needs of a particular user community, which the general specification cannot afford to do. So user communities are encouraged to define

their own profile in case the main body of the specification is not sufficient to ensure interoperability. Such profiles will in turn serve as guidance for developers and product vendors who can claim compliance with these.

4.5Mistake #5: The composition of two Specifications is just the Sum of these

Specifications refer to each other, and compose with each other. For example, a protocol layer will operate above another protocol that is defined in another specification. Or, some interface used in combination with another interface. Often the specification writer wishfully assumes a ?black-box? or perfectly modular composition, and ignores the ?corner cases? (error escalation, mismatched features) leaving the details to implementers. The degree of variability introduced by such compositions is then underestimated. As a result, implementers may interpret differently how the composition is supposed to operate.

The composition of implementations of these specifications is a separate issue that will depend on engineering choices - but still predicated on how precisely the composition of these specifications has been specified.

Best Practice:

- For specification writers: When a specification S1 ?uses? another specification S2 with the intent of not depending excessively on S2 (i.e. keeping the possibility to replace S2 with S3 later on), it is good practice to gather in a single section everything that concerns the details of composing S1-S2. This is the case of ?Bindings? sections often found in appendix. This requires the addition of new error types, or the capability to ?escalate? or interpret errors from the other specification. Such composition details could also be defined in subsequent, separate profiles. In any case, it should be expected that new bindings or profiles will be added over time, so it is wise to not overly tie the specification of the composition itself with the main specification body.

4.6Mistake #6: Extensibility: NIMSY (not in my specification yard)

Extensibility makes it possible to extend either the specified artifact or the functions of a specified device. A common mistake is for specification writers to treat extension points in their specification as customization hooks that are fully out of scope and the usage of which they have no responsibility for. As a result, these extension points may be used and interfere with the core specification in ways that were not foreseen, and cause unexpected interoperability problems.

Best Practice:

- While extensions are out of scope of a core specification, how to handle failures to support an extension is not. How to gain awareness of extensions used by a partner implementation must also be considered by specification writers. How an implementation is expected to react to an unrecognized extension must also be described (which errors are generated?) - for example, the*mustUnderstand* attribute in a SOAP messaging header dictates the expected behavior of a receiver. If extension points should not /cannot be used to override the default feature or behavior, this should be clearly stated. If it can override it, the specified feature should be worded to allow so (e.g. "...unless extension XYZ is used to indicate otherwise"). It is good practice to ?flag? every extensibility point in a specification, e.g. in an appendix, as a possible interoperability hazard. The same precaution as for optional features (see #4) apply.

4.7Mistake #7: Backward Compatibility: it is all or nothing (and if it is not all, let us just not talk about it)

- Backward compatibility: A standard is said to allow backward compatibility, if products designed for the new standard can receive, read, view or process older standards or formats. Or, it is able to fully take the place of an older product, by inter-operating with products that were designed for the older product.
- Forward compatibility (sometimes confused with extensibility) is the ability of a system to gracefully accept input intended for later versions of itself. The introduction of a forward compatible technology implies that old devices partly can understand data generated by new devices.

Assume your new specification version is not backward compatible ? in general, an embarrassing fact you'd prefer users to not ask about. Yet the worse thing to do is to avoid talking about it, hoping no one will notice ? which may indeed be the case in the very unlikely event where everyone migrates at the same time to the new version. Users will implicitly assume backward interoperability, and they are on their way to be disappointed and angered.

Best Practice:

- Describe precisely what are the non backward compatible features, and on the public relation side, explain in an FAQ why it wasn?t possible to preserve compatibility. A section describing the "diff" from V(n) to V(n+1) will greatly help implementers of the new version to understand what they can and cannot reuse from their implementation of the previous version. It may help also to specify how features of a previous version map to - have been replaced by - similar features in the new version. Now, some subset of features may still be backward compatible if not all features, and these should be identified. That will greatly help users who may actually be only concerned with these features, as well as developers who will try to architect a multi-version implementation. Describe the expected effect of using the "wrong" version when the new one (or the old one in the forward compatibility case) is handled. You might be able to define a related (and restricted) conformance profile for which backward compatibility is supported. The whole point is to honestly help users understand what can and cannot be reused from a previous implementation or environment.

4.8Mistake #8: Writing for an Exclusive Club

Who are the readers of your specification? If this is only the club of experts who spent 2 years together coming up with a consensus on how to do business in a specific area, then the specification does not need be more than a glorified MOU: every word conveys a well understood meaning in your club that does not need be further detailed, given the common background and domain culture of the authors (the club members). But if the

specification must be read by implementers, end-users, test developers or product developers who may not fully share the author?s background and expertise, then interpretation mistakes or liberties will abound.

Best Practice:

- It is best to assume that your readership will have different backgrounds. Think of the graduate intern who may have to implement it and does not have your domain culture. This will be even more likely if your standard is widely published, adopted and translated in a foreign language. Being too explicit or repetitive does not hurt - this is not the same thing as being too verbose. Define a glossary of terms used, that gives them precise meaning in the context of this specification. Do not hesitate to "steal" terminology definitions from elsewhere, and repeat it. Do not hesitate to redefine common terms used with a specific meaning in your context. For example, avoid vague statements such as "Processor X must "accept" an artifact Y with feature Z" , unless "accept" is defined in your glossary. If some unspecified features or behaviors are delegated to referred specifications, state this explicitly and don?t assume this is obvious to the readers. If you use terminology borrowed from these external specification, add a mention about it, e.g. "in the sense given in [...]".

4.9Mistake #9: The Conformance Clause will fix all that

Conformance clauses are not an excuse to keep a the main body of a specification wide-open to interpretations, too vague or simply incomplete. Future conformance clauses will multiply these interpretations and jeopardize interoperability. Pressure to complete a specification may lead to such abuse, with a premature release of the specification leaving the publishing of the (overloaded) conformance clause to a later date. This is enough to confuse implementers who assume wrongly that the specification alone is sufficient for their implementations to be interoperable.

Best Practice:

- The conformance clause - now required in all OASIS specifications - is not to be used as additional specification material, but is an interpretation of it in terms of conformance. Its writing style should leave no doubt that it is not a continuation of the main specification body. The main body of a specification should be as precise as possible, and remain the main reference for developers, understandable and coherent on its own. At least one conformance clause should be published at the same time as the specification (a mandatory requirement in OASIS specifications). It is recommended here that a "main" or "basic" conformance clause be defined for every specification, that covers the most basic usage(s). The conformance clause must identify which type of implementation(s) it addresses - in case different entities are concerned. For example, a conformance clause may restrict itself to "message receivers" in a messaging standard, while another clause may concern both senders and receivers but with a restricted yet compatible set of functionalities.

4.10Mistake #10: Our job is only to Specify. Verifiability is the job of Test Suite writers

Sure, but if you do not provide at least hints on how some specification requirements might be verified, test suite writers will interpret your statements as they wish, or may just ignore them. It will not be until systems are in production, that conformance discrepancies - and therefore interoperability problems - are discovered. If there is no way to test some requirements, then conformance and interoperability hangs solely on the interpretation of

your words by implementers.

Best Practice:

- Ideally, the "testing perspective" should be added to a specification in the form of a list of test assertions. These could be provided in a companion document.

Dates

Approved:
Sat, 2012-01-14
Effective:
Sat, 2012-01-14