Interoperability

iov42 Limited
Regina House
124 Finchley Road
London NW3 5JS
United Kingdom

Table of Contents

Interoperability

Introduction

Interoperability in the blockchain space is often loosely defined as the ability for differing technologies to interoperate with each other, often in such a way that events in one blockchain trigger events in another.

In the world of Distributed Ledger Technologies (DLT), interoperability has become a buzzword which is at risk of becoming a solution looking for a problem. It is always inevitable when we consider that any new significant paradigm shifting technology tends to bring a plethora of competing approaches, protocols and implementations.

However, these new approaches often bring value to slightly different problem spaces, business domains and use cases. So it's unsurprising that at some point someone declares that: *wouldn't it be great if these different solutions to different problems could just interoperate with one another.* This abstract notion in and of itself isn't a bad thing, but can become so when it is done for the sake of it with no clear purpose or desired outcome.

Any discussion of interoperability in the abstract, without grounding it in practical use cases, although interesting, runs the risk of being largely academic.

At this point in the development of the DLT landscape, interoperability has become something of a panacea that can get a technology immediately dismissed unless the answer is a resounding "yes we support interoperability". Unfortunately such black and white approaches can miss the important nuances in what interoperability is and why you might want to invest in it.

Why interoperate?

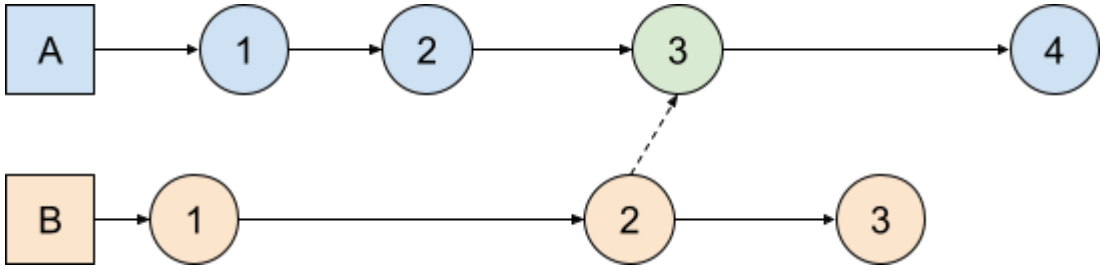
The goal of a DLT is to ensure a single source of truth with regard to events that occur. The 'what' and the 'when' are of supreme importance. It could be argued that in an ideal world, there would be a single protocol that allows anything and everything to record what matters in a secure and cost effective manner. The technology space that has emerged is highly fragmented.

If we consider two hypothetical DLTs, call them A and B, then we can begin to consider modes of interoperability and even speculate as to example use cases that might typify those modes and why you might want to pick one over another. We'll use this analysis technique to look at the different approaches to interoperability as we progress forward.

The base assumption of the following modes is that both DLTs provide an immutability guarantee, a sense of finality at some point in which each DLT can expect that the history recorded in either cannot change for any reason. If such things as hard forks that rollback transaction state can occur, then meaningful interoperability arguably cannot, as a rollback on one DLT would imply a rollback on the other. This seems like a significant engineering challenge and so interoperability here shall be considered only within the guarantee of immutability.

Read-interop

DLT A builds a use case in which an event occurring on DLT B triggers a state update in DLT A. In this scenario DLT B needs not even be aware of DLT A.



As you can see in this diagram, the event '2' in DLT B causes event '3' to be created in DLT A.

In order for such a pattern to be realised, DLT A must be able to read the changes occurring in DLT B. This is straightforward with public blockchain technologies as the data can simply be read by running a node in the network or even by reading from a public blockchain explorer.

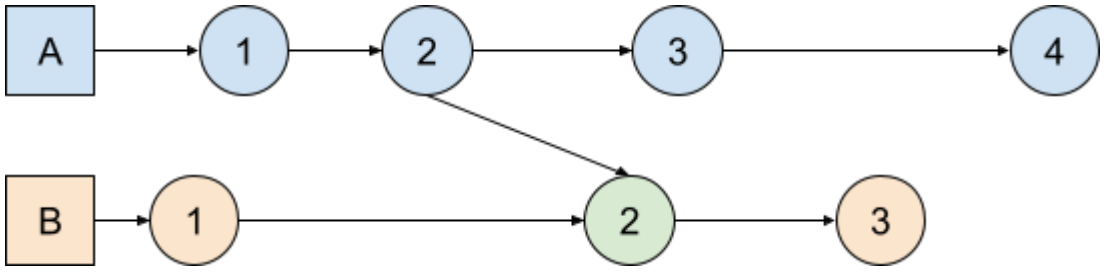
Now we should consider if this pattern is in fact useful for any conceivable use case. A possible example is one in which purchases of a commodity represented on DLT B are immediately tokenised on DLT A. For example, DLT B could store events relating to purchases of a precious metal such as gold. Users of DLT A could make purchases of quantities of gold on DLT B that are visible to DLT A. Once such a purchase is made, DLT A could eventually create tokens in response to the gold purchased on DLT B. These tokens would appear *eventually* and not exactly at the time of the purchase on DLT B, it entirely depends on the details of the constraints of the use case, but this level of guarantee may be sufficient, it may also may not. However, it's extremely important to note that **eventual consistency is not acceptable for many categories of solution** as will be discussed shortly.

This pattern is most suited where there's a desire to mirror some event from one DLT to another, the mirrored event could be identical or different, mirroring here means that an event in one DLT eventually triggers some event in another. In the example of gold tokens above, eventual consistency would be insufficient if there's any possibility of the underlying gold asset on DLT A being sold before it's reflected on DLT B, we could have a situation in which the two DLTs differ around the ownership and this could be a problem, it entirely depends on the use case.

There's some deliberate simplification here with regard to linking users present within both DLTs, but such considerations are not the purpose of this document.

Write-interop

DLT A builds a use case in which an event occurring in DLT A causes DLT A to create a corresponding event in DLT B. In this scenario both DLTs are aware of each other in some meaningful way, specifically in that DLT A is able to write directly to DLT B.



The diagram shows that event '2' occurring in DLT A causes DLT A to create a corresponding event in DLT B. The event could be equivalent or entirely different and the numbers here do not imply equality.

To achieve this pattern some mechanism needs to exist within DLT A that allows it to submit a transaction to DLT B. This could be achieved with a sufficiently powerful smart contract or some sort of middleware technology. It's important to note that DLT A will likely have no control over when the corresponding event in DLT B will become visible or achieve finality and so is also an *eventually consistent* model.

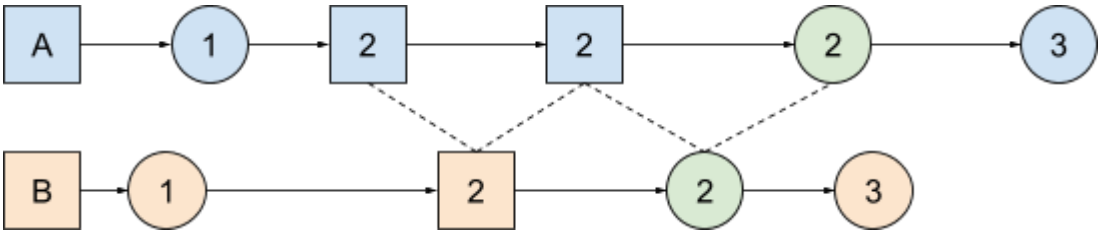
This pattern can be used to achieve the prior use case of gold tokenisation, just in a different way. If we consider that gold is now present on DLT A and the tokenisation occurs on DLT B then we can see how to implement it. The subtle difference in the read or write interop solutions comes down to where you can put the business logic to do the tokenisation; whether it's on the read side or the write side of the event.

As with the previous pattern, this approach is also suitable for the mirroring of events.

A further use-case to consider is one in which a value movement on DLT A creates a corresponding value movement on DLT B. For example, a transfer of a cryptocurrency on DLT A could trigger the transfer of an NFT on DLT B. The downside of using this approach for such a use case is that the value movement on DLT B is eventual and in fact might occur with a potentially unbounded delay, i.e. never. Few users would be comfortable performing a financial transaction in which they might never receive their goods or services! A better solution is covered in the next approach.

Atomic-interop

This could also be referred to as full-interop or perhaps read/write-interop. The goal in this model is that both DLTs will expose dependent events at approximately the same time. DLT A would create an event that's not 'visible' or 'committed' until DLT B has its corresponding event.



In this example, DLT A announces an event '2'. The square shows that this event isn't accepted yet, it's just in a 'prepare' phase. DLT B receives this event, whether by the read or write model described previously doesn't particularly matter. DLT B acknowledges the event by creating a corresponding entry on its ledger. DLT A on seeing the acknowledgement can either commit or rollback the event, in this example it chooses to commit, denoted by the corresponding square. DLT B sees this commit instruction and creates the full committed event '2' (the circle). At this point no rollback can occur and DLT A must eventually commit event '2' as it does in the diagram. This is of course a classic 2 phase commit protocol.

Such a protocol is expensive in terms of the number of distinct messages that must occur in order to create events with a guarantee that they appear on both DLTs. This is why it's important to consider if a use-case needs a fully atomic protocol such as this or whether the weaker guarantees above are sufficient.

A good example of a use-case that would necessitate this is any classic financial protocol, a topical example being Central Bank Digital Currencies (CBDCs) and how a cross DLT atomic swap might happen. DLT A might host a CBDC called A-COINs and DLT B B-COINS. An example swap could be one in which DLT-A proposes a swap of 100 A-COINs for 200 B-COINs.

On receiving the proposal, DLT B (if it agrees) would create the acknowledgement event on its DLT. If a timeout occurred before this, then DLT A could rollback the request. Once DLT A acknowledges the acknowledgment then the guarantee is that DLT B will eventually commit.

This mechanism makes it possible to migrate a token from one DLT to another. Once the protocol has begun, an external observer must assume the token is locked. Which system it belongs to at this given point is in flux. The protocol either completes or rolls back, but at no point during the protocol can the token be considered to belong to either DLT. Unlike the previous patterns which are suitable for mirroring events between DLTs, this approach is suitable where you'd like an update to appear to happen on both DLTs atomically.

Summary

Approach	Consistency	Cost in messages	Suitable for
Read-interop	Eventual	1	Mirroring
Write-interop	Eventual	1	Updating
Atomic-interop	Strong	4	Financial

The iov42 DLT can perform either read or write interop using middleware technology. I.e. a simple process that watches external DLTs and makes updates to the iov42 DLT or watches the iov42 DLT and makes updates to external DLTs. In the future, Smart Assets (which are our approach to smart contracts) may be able to carry this burden without using middleware, but there are complex issues to be considered, such as what to do if the external blockchain isn't available at the time of request. A system of retry/recovery would need to be designed and engineered to safely achieve this.

In order to achieve atomic-interop a new set of APIs would need to be considered that expose a 2 or 3 phase atomic commit protocol. Such APIs would also need to be present in some manner on external DLTs in order for them to interact and so a standard would need to be designed to facilitate this, which is no small undertaking.

Simpler solutions could be envisaged where smart contracts on the external DLT are used to implement an atomic commit protocol. An evolution of an ERC20 token on the Ethereum network could provide the necessary semantics to move tokens between platforms. The smart contract could use the 'self destruct' primitive once the protocol completes.

Unfortunately it seems that the engineering effort to achieve a standard that could be used for full atomic-interop between competing DLTs might be significant and require a consortium of actors in the DLT space to agree. It's also clear that although laudable, the goal of such interoperability would need to generate significant value for both interacting DLTs for it to be worthwhile.

Conclusion

Discussing the interoperability of DLTs without discussing it from the point of view of a specific use case is potentially unhelpful in that it ignores the nuance and complexity of the actual underlying approach required.

Incentives need to be strongly aligned between competing DLT technologies in order for them to make the necessary investment in exposing an atomic commit protocol. Indeed a standard would need to emerge for this to be the case and it's not clear that strong enough motivators exist in the market for this to happen.

It is our opinion that the concept of interoperability is too important for its discussion to be had in the abstract, where it risks being perceived as a meaningless buzzword.

The iov42 blockchain can achieve both read and write models of interoperability. The iov42 network model is of a small number of Proof of Authority nodes based on scalable data centre technology, coming together to form a "Zone" that allows enterprise and government to develop use cases for customers and citizens.

It has always been central to the iov42 thesis that we'd develop the necessary machinery required to allow these zones to communicate with one another, primarily to allow for use cases to overlap and interact where individuals have need or desire.

For this reason we plan to develop interoperability between our own zones in the first instance that might form the basis of a standard that other DLTs could adopt should they wish to do so, if use cases are identified and the relevant incentives are aligned.

The desire and need for use cases to overlap and interact has been part of iov42's thesis from the very start, informing our concept of zones and driving the design of their communication machinery.

Standardised or not, "interoperability" as it is understood between DLTs will result naturally from our own zone interoperability implementation if and when use cases are identified and relevant incentives align.