UK Finance

EY
Building a better working world

Technical Report

UK Regulated Liability Network Experimentation Phase

September 2024

Contents

List of Tables

List of Figures

Disclaimer

This document reflects the views of UK Finance Limited (UK Finance) and sets out the findings following the UK Finance Regulated Liability Network Experimentation Phase (2024). It is aimed at building on the previous work done on the Regulated Liability Network original white paper published in 2022 and the 2023 UK Finance Regulated Liability Network Discovery Phase.

Please note that this document is intended to provide general information only. It does not represent legal, financial, investment, tax, regulatory, business, or other professional advice and should not be relied on as or instead of professional advice. Nothing in this document constitutes a waiver of legal privilege by UK Finance or participating members. Neither UK Finance nor its participating members represents or warrants the completeness or accuracy of the information within the document. Nothing in this document shall operate to be binding on UK Finance or it's its participating members, nor does this document give rise to any enforceable obligations or duties on UK Finance or its participating members. Please note, this report does not reflect the views or official positions of the participating members.

UK Finance, and any of their participating members, officers, employees or agents, shall not be responsible or liable to any person for any loss, damages or costs arising from or in connection with any use of the document or any information or views contained herein. UK Finance reserves the right to edit, amend or withdraw this document without any reference to users. Users of the document should ensure that it is suitable for their use and that appropriate due diligence has been conducted, including in relation to compliance with relevant applicable laws.

UK Finance and its participating members were contributors to the UK Finance Regulated Liability Network Experimentation Phase (2024) and this document. However, participating members were not the authors of the document. Please note, this document does not reflect the official views or positions of the participating members. Users wishing to use or share this document shall credit UK Finance, and where applicable, the vendor(s) as the authors of the document.

01

Executive summary

Executive summary

This report describes the results of the technology workstream of the Experimentation Phase of a project to design, implement and test a concept for a new Financial Market Infrastructure and a 'platform for innovation' in the UK. By delivering a new multi-asset, multi-issuer programmable tokenisation platform, and an orchestration service and common APIs, this infrastructure could provide a number of foundational capabilities, including interoperability and programmable payments, across all forms of money, in a way that provides functional consistency[1] and maintains the singleness of regulated money.

This report provides an overview of the overall vision for the project and the context in which it was devised and executed. This is followed by a summary of the scope, and analysis of the requirements, as well as a summary of the key findings. The solution is then described. Finally, we make recommendations for future phases. A full exploration of the design decisions underpinning the project are outlined in the Appendix.

The Experimentation Phase delivered an extensible platform for innovation comprising:

- An API and Orchestration Layer[2], enabling interaction with all forms of money and a variety of new and existing ledgers, delivering functional consistency, programmability and access to a rich set of 'foundational capabilities'[3].
- A multi-issuer Tokenisation Platform[4] that facilitated the issuance of tokenised commercial bank deposits, as well as tokenised retail and wholesale Central Bank Digital Currency, with programmability and in a way that met the required privacy goals.

This was supported by the implementation of:

- A suite of Business Applications[5] to test the API and Tokenisation Platform, and to demonstrate and validate the Foundational Capabilities across forms of money, including existing commercial bank money and tokenised deposits.
- Integrations with simulations and prototypes of Open Banking, Project Rosalind, and a simulated Real-Time Gross Settlement (RTGS) platform[6], with associated test data to facilitate exploration of the various combinations of sources and destinations for payments this allows, including business flows to convert between forms of money or between ledgers.

1 Functional consistency is the principle that different forms of money have the same operational characteristics. It is described in this paper: https://arxiv.org/pdf/2308.08362

2 The API and Orchestration Layer is described in more detail from page 38.

3 See page 26 for more information on the Foundational Capabilities concept.

4 In this report, we refer to this layer in terms of its function and purpose – a multi-asset, multi-issuer, programmable tokenisation platform – rather than in terms of a particular technical implementation, such as a DLT, shared ledger, or network of networks. The Tokenisation Platform is described in more detail from page 27.

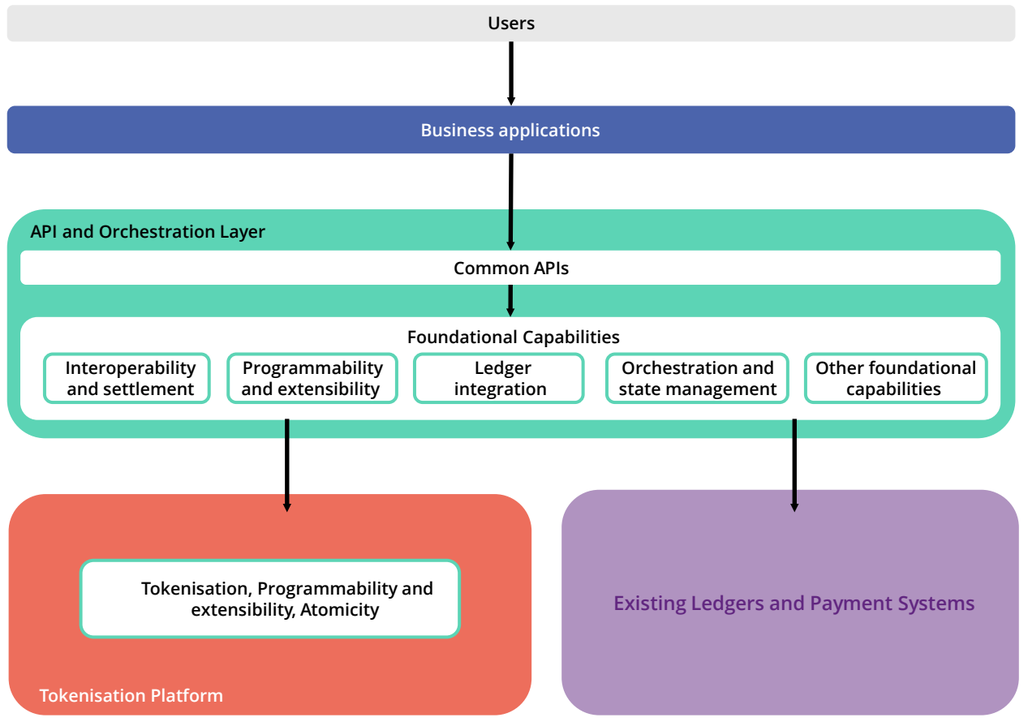
5 See page 28.

6 See page 32.

The overall high-level architecture of the Experimentation Phase is depicted in Figure 1, below, which shows the API and Orchestration Layer (green), and the Tokenisation Platform (orange).

The supporting Business Applications are shown at the top (blue), interacting with the API, and the simulated ledgers are shown at the bottom right (pink).

Figure 1. The Experimentation Platform consisted of an API and Orchestration Layer (green), and a multi-issuer programmable Tokenisation Platform (orange). A suite of Business Applications (blue) and simulated ledgers (purple) were delivered to support testing and facilitate experimentation.



A common API and Orchestration Layer across all forms of regulated money[7], and a multi-issuer programmable Tokenisation Platform into which different forms of money can be issued, were both found to be technically feasible.

API and Orchestration Layer

- The project implemented a set of fourteen 'Foundational Capabilities', and five 'Business Applications' that utilised the Experimentation Platform to achieve a wide array of operations interoperating across different forms of money.

7 And across all types of ledger.

- The orchestration layer demonstrated that functional consistency across all forms of money is an achievable aim. The project showed that capabilities of existing forms of money could be upgraded to match novel features of other forms, and did so without requiring fundamental changes to the form or type of money that is issued by any individual institution.
- An orchestration layer is a potential enabler of innovation, delivering interoperability and facilitation of synchronised settlement[8] using timelocks.
- Interoperability across various forms of money, including settlement in central bank money as well as across the UK Faster Payment System (FPS)[9], was shown to be possible and can maintain the singleness of money and the role of wholesale central bank money as an anchor for trust and confidence.
- Programmability could be a viable way to provide extensibility to the payments system in an efficient and controlled manner. This could be validated in a subsequent hackathon.

Tokenisation Platform

- Issuance of tokenised deposits and tokenised Central Bank Digital Currency (CBDC) was simulated, and a set of transactions performed, demonstrating programmability and the potential for atomic settlement[10]. Delivery-versus-Payment was explored, and this was tested using three different accounting and integration models for tokenisation (direct, indirect pooled, and indirect shadow[11]).
- The Tokenisation Platform design included consideration of situations where fully pooled governance and a fully shared ledger cannot be assumed, contributing to the broader industry exploration around the development of multi-national, multi-asset, multi-issuer, interoperable programmable ledgers[12].
- The project demonstrated multiple models for the issuance of retail CBDC as well as the use of tokenised wholesale CBDC to facilitate settlement of tokenised commercial bank deposit transactions. Future work could explore the potential for a retail CBDC to be issued to a common platform alongside commercial bank money, as well as the possibility of a potential new 24/7 instant settlement capability facilitated through tokenised wCBDC.
- The potential of some shared ledger architectures to enable simultaneous cross-partition atomicity could be of particular value in wholesale use-cases and suggests that further work to evaluate the use of a shared ledger for such scenarios could be beneficial.

To achieve its aims, the project built and tested twenty-four flows across five forms of money (existing wholesale central bank money, Tokenised Deposits, regular commercial bank accounts, Retail CBDC and Wholesale CBDC, including tokenised forms), and analysed dozens more flows. A significant amount of functionality was delivered and tested in a short space of time, supporting the hypothesis that an architecture of this nature can be built relatively quickly and efficiently.

8 Where a set of updates are applied to a single database or ledger system and can be confirmed as a single technical operation we refer to the process as 'atomic'. Where two or more distinct systems are involved, a process of coordination is required to ensure updates occur in an all-or-nothing fashion across the separate systems. We refer to that process as 'synchronisation'.

9 Initiated via Open Banking.

10 Atomic settlement across the Tokenisation Platform would depend on the issuance of tokenised wCBDC and an architecture with a shared consensus provider (eg a single notary in Corda-based designs).

11 See page 60 for an explanation of these terms.

12 e.g. the Unified Ledger initiative of the Bank of International Settlements: https://www.bis.org/publ/arpdf/ar2023e3.htm

The project identified several other critical learnings, which can inform future work:

- The suite of Foundational Capabilities proved suitable for third-parties building sample Business Applications. For example, card integration with merchant terminals was demonstrated. Latency was not tested so future work should validate that the architecture can support the latency required by card networks given their particular requirements in this regard[13]. The business application vendors identified specific APIs that would have made their work easier. Obtaining this feedback was a key objective for the project.
- The core 'Functional Consistency' principle, that it is possible for different forms of money to have the same operational characteristics, was demonstrated. For example, some use-cases require a concept akin to locking, but not all forms of money support this. The project demonstrated how different implementations of the concept could be provided for different forms of money, such as a model based on escrow, with only limited differences of behaviour being visible in the API.
- The likely requirements for the orchestration layer to support reconciliation with banks' existing accounting systems were identified, and the relative advantages and disadvantages of different forms of settlement (synchronised and atomic) were studied. In particular, multiple tokenisation models were explored, and their bank integration impacts (such as with deposit systems, reconciliation processes and customer statements and advices) were examined.
- A federated 'network of networks'[14] model was explored for the tokenisation platform. This highlighted the potential to support retail-scale transaction volumes and showcased an architecture that could allow a Tokenisation Platform to be created incrementally, federating pre-existing single-issuer networks over time. This maximised the autonomy and control of each issuer but made it more difficult at a practical level to directly showcase some core shared ledger differentiators including simultaneous atomic settlement across partitions and some models of programmability.

13 Note that testing of non-functional requirements was in general out of scope for this project

14 This is an architecture in which the Tokenisation Platform is built on a single technology, Corda in this case, but where each tokenised deposit issuer had their own 'application network' on which their own customers were modelled. The design, and the decisions underpinning it, are described in the Architectural Decisions appendix, from page 79.

02

Vision and context

Vision and context

The payments landscape in the UK is changing. A retail CBDC – the Digital Pound – is being actively researched, phenomena such as stablecoins are emerging, and regulatorily compatible distributed ledger platforms have matured.

The Regulated Liability Network concept has set out a vision for how this technology can be used to deliver a generational change in how assets of all types are managed and transacted.

The UK RLN project is an attempt to chart a path forward for the UK that tests the RLN concept in a domestic retail context: preserving the 'singleness' of the pound and preventing fragmentation in retail payments and deposits, while still ensuring wholesale use cases can be addressed.

In so doing, the project's vision is to enable a 'platform for innovation' for money in the UK.

At the heart of the project's technical vision, therefore, are two distinct but complementary concepts:

1) A Tokenisation Platform with the ability to support the tokenisation of multiple types of assets[15], including commercial bank money and CBDC, by multiple issuers in a way that supports programmability and the potential for atomic settlement, with strict privacy[16].

2) An API and Orchestration Layer that presents a rich API across all forms of regulated money, in a manner that promotes innovation and is capable of wide adoption. This programmable layer provides a set of foundational capabilities that support functional consistency[17] across the digital pound and various forms of commercial bank money including tokenised deposits.

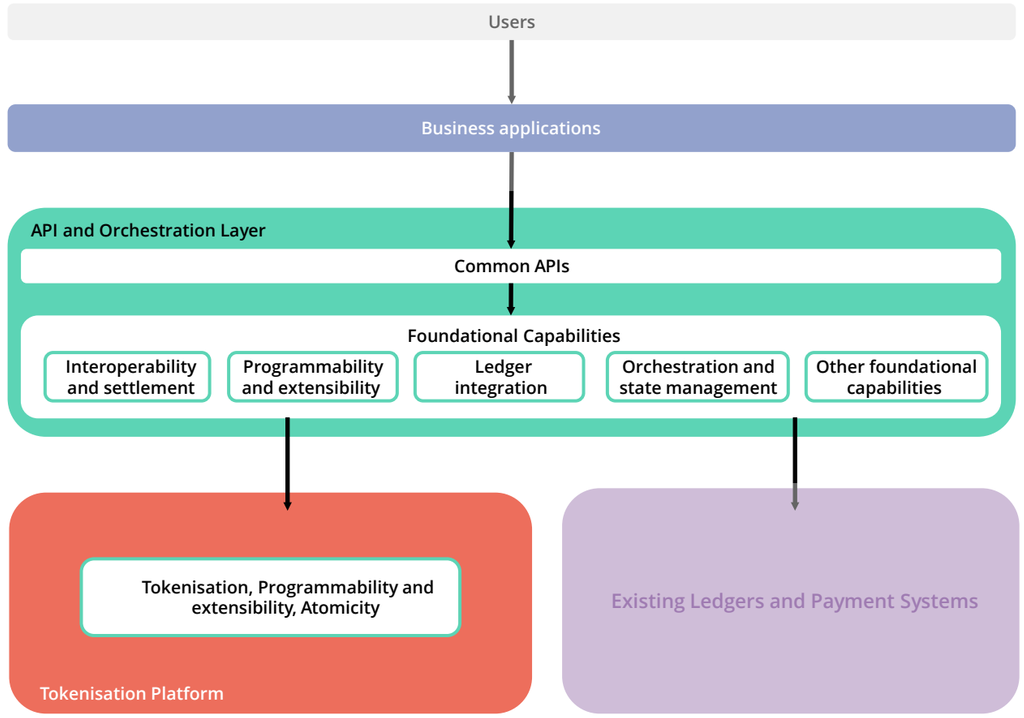
15 In this report, we often use the word 'asset' as convenient shorthand for 'a record managed by the Tokenisation Platform, even though from the perspective of the issuer, a tokenised deposit is a liability, and not an asset. Similarly, the use of the word 'asset' is not intended to express a view on how these records are viewed legally.

16 https://regulatedliabilitynetwork.org/wp-content/uploads/2022/11/The-Regulated-Liability-Network-Whitepaper.pdf

17 https://doi.org/10.21314/JFMI.2023.003

These two components are shown graphically in Figure 2, below.

Figure 2. The primary concepts in the project were the Tokenisation Platform (orange), and the API and Orchestration Layer (green). Other components of the project greyed out for clarity.



The key objectives of the Technology Stream of this Experimentation Phase of the project were to:

- Demonstrate a 'platform for innovation' operating across multiple forms of regulated money and multiple types of ledger.
- Demonstrate tokenisation of commercial bank deposits and central bank money on a common platform.
- Build a working proof of concept, with a view to demonstrating technical feasibility.
- Successfully test Foundational Capabilities and deliver use cases such as home buying and payment upon delivery of a physical product[18] that exploit and test these capabilities.
- Demonstrate a platform that can provide for both retail and wholesale scenarios.
- Provide evidence that specific non-functional requirements such as data privacy could be met.
- Provide an inclusive vision for all forms of digital money that reduces fragmentation risk.
- Facilitate legal and business analysis.

18 https://www.ukfinance.org.uk/news-and-insight/press-release/uk-finance-announces-new-regulated-liability-network-experimentation

Conceptual Framework

As described above, the UK RLN project is best understood as delivering two distinct, but complementary, components:

- an API and Orchestration layer, and
- a DLT-based Tokenisation Platform.

In practice, these two components are tightly linked in the design and implementation, but are considered separately here for the purposes of explaining the conceptual framework.

API and Orchestration Layer

The API and Orchestration Layer is intended to explore whether it is possible to define and build a new API across all forms of money that promotes innovation and delivers 'functional consistency'.

In other words, can we provide a rich set of foundational capabilities that sit 'above' existing and new forms of money (commercial bank deposits, a putative retail CBDC, tokenised deposits and so forth) which allow third parties to deliver innovative new solutions? And can this be achieved without developers concerning themselves with the underlying type of money or ledger while allowing them to exploit as many of the unique features of the underlying platforms as possible? Example of features that only a subset of the underlying platforms natively support might be certain types of 'locking' or 'programmability'.

One role of the orchestration layer[19] is to support a consistent API across the underlying implementations while exposing features that only some of those implementations can provide. Where an underlying platform provides a feature, the orchestration layer *delegates* to it. Where a platform does not provide a feature, the orchestration layer attempts to implement a reasonable *substitute*.

Locking is an an example of a feature that not all forms of money provide, and escrow is an example of a reasonable substitute in some situations. One of the objectives of this project was to evaluate the extent to which this approach is achievable and valuable.

A useful way to think about the API and Orchestration Layer is as an enabler of a 'platform for innovation', providing a single entry point for applications wishing to interact with any and all forms of regulated money in the UK, with responsibility for facilitating the routing of payment messages, coordinating the movement of money between parties irrespective of where the money starts and ends, and implementing new capabilities where the underlying systems do not have native support[20]. It would do so with a level of security, scalability and other non-functional characteristics expected of an infrastructure of this ambition and sensitivity, and it would expose a programming model, enabling its capabilities to be extended over time.

19 Where the context permits we use 'orchestration layer' as shorthand for the API and Orchestration Layer.

20 An example of another enabler of innovation would be the potential of a Tokenisation Platform (next section) to enable tokenised forms of money.

The themes and functions exposed by the API and Orchestration Layer, such as simple push payments, interoperability across forms of money and across ledger types, programmability and so forth are collectively described as *Foundational Capabilities*[21]. A parallel strand of activity in this project was to build a series of indicative *Business Applications*[22] that utilised and explored those capabilities.

Tokenisation Platform

The second theme of this project was the design and implementation of a tokenisation platform[23]. This maps directly to the broader Regulated Liability Network concept: a common technology infrastructure to which banks (commercial banks as well as the central bank) can connect for the purposes of projecting portions of their balance sheets or custody databases. It thus has the potential to enable a multi-asset, multi-issuer, programmable ledger that can facilitate atomic settlement[24] and potentially enable new settlement venues.

As the RLN whitepaper explains, the concept creates the "potential for a regulated Financial Market Infrastructure (FMI) that could deliver an interoperable network of all facets of the sovereign currency system: central bank money, commercial bank money, and e-money (and in the future, regulated stablecoins)."

Going further, "the purpose of RLN would be to create a new shared ledger substrate for the sovereign currency system that is 'always on,' 'programmable,' and 'multi-asset.' The network would deliver 'on-chain' finality of settlement between the participating institutions in sovereign currencies and be compliant with all existing rules and regulations."

Support for retail and wholesale CBDC issuance, with settlement in the latter as an option, was in scope.

In the Experimentation Phase, a 'maximally autonomous' design was implemented, and formed the basis of the experiment. This was shown to meet the project's functional requirements. However, it is possible that this came at the expense of some of the purported benefits of a more fully-shared architecture. In particular, settlement of transactions involving multiple different assets or issuers requires the synchronisation of two or more technical components when using the network-of-networks model, whereas a fully shared ledger may be able to achieve this more simply, as an atomic event.

21 Some of these capabilities are provided by one or more underlying ledgers, such as Foundational Capability 8: Tokenisation of commercial bank money, which is implemented solely within the Tokenisation Platform. However, all capabilities are accessible via the API and Orchestration Layer.

22 Described from page 40

23 We use this phrase to capture its *purpose* rather than its *form* (e.g. distributed or shared ledger, etc), as the latter was itself a topic of discussion in the project

24 Where tokenised wCBDC is available and with a shared consensus design.

03

Scope and Requirements

Scope and Requirements

The primary technical deliverables of the project were:

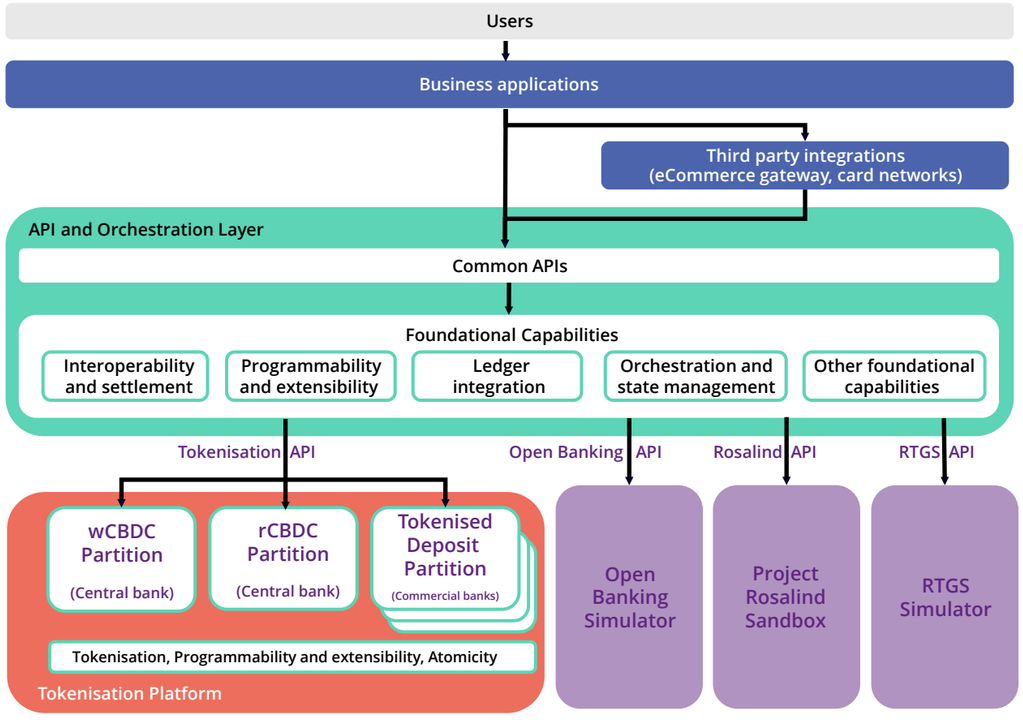
- An API and Orchestration Layer; and
- A Tokenisation Platform

These were supported by two additional deliverables:

- A set of simulated ledgers, to which the Orchestration Layer connected, and
- A suite of Business Applications to test the API and Tokenisation Platform, and demonstrate and validate the Foundational Capabilities.

Graphically, these deliverable are depicted in Figure 3, below.

Figure 3. The project's primary deliverables were the API and Orchestration Layer (middle, green), and the Tokenisation Platform (bottom left, orange). These were supported by simulated ledgers (bottom right, purple) and Business Applications (top, blue)



To enable testing, a large set of test data was defined and, in some cases, the same functional requirement (such as support for retail CBDC or tokenised deposits) was implemented in several different ways so that the differences and relative merits could be explored. As such, the design of the Experimentation Phase differs in some key respects from how a production system might be built. We explain where this has significance below.

In this chapter, we summarise some of the most significant architectural- or business-specific requirements that informed the design of the Experimentation Platform. We do not exhaustively list every requirement the system needed to meet. Instead, we provide a summary designed to give some insight into the scale and purpose of the system that was implemented.

API and Orchestration Layer

The purpose of the API and Orchestration Layer is to provide a single secure API 'entry point' that enables functional consistency across all forms of money, and does so in a way that promotes innovation.

The responsibility of the API was to provide a secure set of endpoints that third-party applications could use to interact with the Orchestration Layer and the ledgers that sit underneath it.

The API and Orchestration Layer's responsibilities were to:

- route requests to the appropriate underlying ledger or ledgers, aggregating responses where required
- coordinate the orchestration of operations between ledgers and forms of money, including settlement
- simulate or otherwise implement capabilities that the API provides but which one or other of the underlying ledgers does not support
- integrate securely with the underlying systems
- facilitate extensibility by the platform's operator to enable subsequent deployment of additional foundational capabilities.

Foundational Capabilities

The initial Foundational Capabilities exposed through the API were:

- Simple push payment
- Interoperability across forms of money. For example, enabling a payer to use central bank money to pay a payee who wishes to receive commercial bank money, and vice versa.
- Interoperability across ledgers. For example, enabling a holder of tokenised deposits to to make a payment to a holder of a regular bank account.
- Payment sequence orchestration. For example, sequencing of payments between multiple accounts such as might facilitate the completion of a house purchase.
- Programmability. For example, the ability to add conditional terms to a payment.
- Locking and unlocking of funds
- Pull payments
- Tokenisation of Commercial Bank Money leveraging the Tokenisation Platform
- Settlement finality
- Settlement through API integration with FPS, Omnibus Account, or RTGS
- Project wCBDC and rCBDC onto the Tokenisation Platform
- Enable external capability. For example, facilitate integration with the existing payment architecture.
- Connector for third-party applications
- Notifications

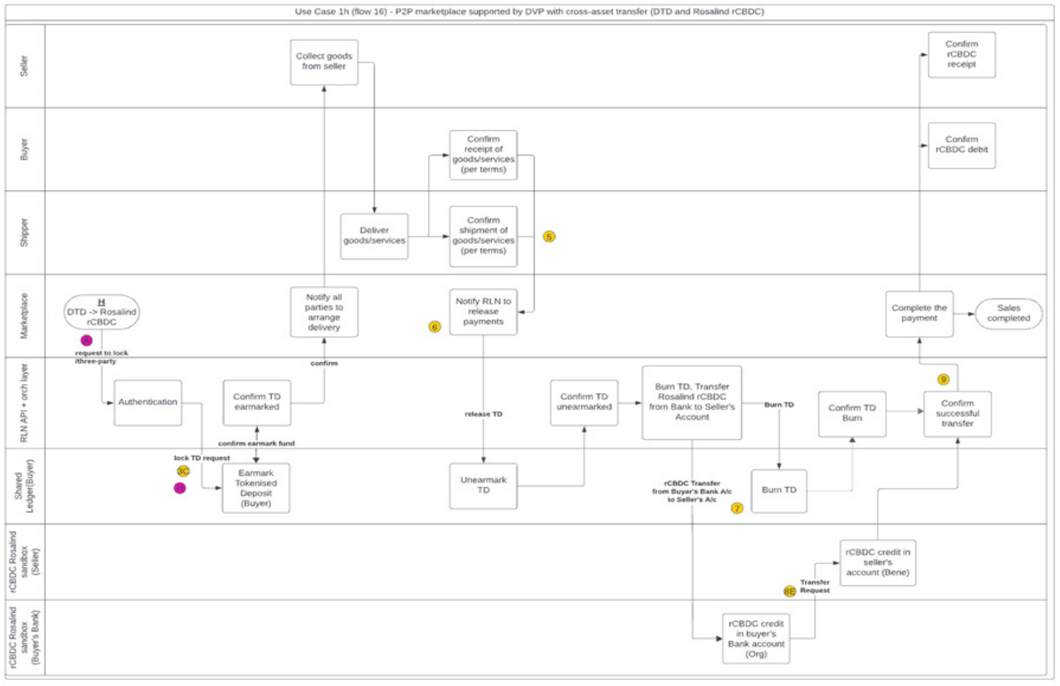
In cases where underlying systems natively supported a capability, the request was routed directly. Otherwise, an appropriate substitute was implemented by the orchestration layer.

Significant Orchestration Flows

To support the above, the API and Orchestration Layer was responsible for implementing a large number of workflows, with associated state management, and integrating with a set of underlying ledgers. For example, one 'simple' push payment may move money from two accounts at the same institution whereas another may move funds between forms of money (such as from commercial bank money to retail CBDC) and across ledger types (such as from a tokenised deposit to Project Rosalind).

The workflows in the Orchestration Layer were required to support this variability and complexity. An example of flow that includes interoperability both of forms of money (commercial bank to central bank) and ledger type (tokenised deposit to Project Rosalind rCBDC) is provided in Figure 4, below. In the figure, we see the steps performed by participants in a 'cash on delivery' scenario (the top four 'swimlanes), and the steps taken by the orchestration layer to coordinate earmarking of funds and transfer of rCBDC (the bottom four swimlanes).

Figure 4. Example of a flow where funds originate as tokenised deposits and recipient receives rCBDC



Tokenisation Platform

The Tokenisation Platform was envisaged as a means of supporting the representation of a range of assets by a range of issuers on a common platform, in a way that enables greater functionality including programmability and atomic settlement, and with privacy.

A key requirement was that each commercial bank be responsible for its own 'partition' and that the tokenisation of their deposits be workable within the UK regulatory framework. The mapping of the 'partition' concept to a specific design was a topic of considerable analysis and debate and is explored in depth in the 'Architectural Decisions' appendix to this report, from page 81.

Furthermore, it was a requirement that two Central Bank partitions be provided, supporting retail and wholesale CBDC, enabling settlement in central bank money to be demonstrated on the tokenised deposit platform and to facilitate comparisons with other mechanisms for the issuance of CBDC.

Programmability and Extensibility

A key requirement of the platform is that it facilitate extensibility by the operator. In particular, the platform must be capable of being augmented with additional foundational capabilities over time, and programmability should be offered at both the Orchestration and Tokenisation layers.

Business Applications

A key hypothesis of the Experimentation Phase is that the proposed architecture could indeed serve as a platform for innovation. Five 'Business Applications', chosen to be illustrative and architecturally significant, described in terms of use-cases, were specified:

1. **Peer-to-Peer ("P2P) marketplace.** The use of locking capabilities to enable a new form of cash on delivery.
2. **Home-buying.** Automating the disbursement of funds upon the completion of a house purchase.
3. **E-commerce merchant gateway.** Using the API and Orchestration Layer to initiate 'pull payments' from any of the underlying ledgers.
4. **Card integration and PoS compatibility.** Enabling merchants to take payments from consumers whose cards are backed e.g. by tokenised deposits.
5. **Settlement of tokenised bond.** Settling bond issuances using the Experimentation Platform.

These business applications were chosen to ensure all fourteen Foundational Capabilities were tested, to inform analysis of specific questions (such as how locking should work), and to generate feedback on the top-level API. In this section of the report, we summarise which Foundational Capabilities were exercised by which Business Application, and briefly discuss the key technical questions they were intended to illuminate.

Foundational Capability Mapping

Table 1, below, sets out which Foundational Capabilities were tested by each Business Application.

Table 1. Each Business Application was designed to test a subset of the Foundational Capabilities

Foundational Capability	Peer-to-Peer ("P2P") Marketplace	Home Buying	E-comm merchant gateway	Card integration and PoS compatibility	Settlement of tokenised bond
Simple Push Payment	✔ (after lock released)	✔ (after lock released)			✔ (after lock released)
Interoperability across forms of money	✔	✔			✔
Interoperability across ledgers					✔
Payment sequence orchestration		✔			
Programmability	✔	✔			✔
Locking/unlocking funds	✔	✔			✔
Pull payment capability	✔	✔	✔	✔	✔
Tokenization of Commercial Bank Money leveraging Tokenisation Platform	✔	✔	✔	✔	✔
Settlement finality	✔	✔	✔		✔
Settlement through API integration with FPS, Omnibus Account or, RTGS	✔	✔	✔		✔
Project wCBDC and rCBDC onto the Tokenisation Platform	✔	✔	✔	✔	✔
Enable external capability			✔	✔	
Connector for third-party application	✔	✔			✔
Notifications	✔	✔	✔	✔	✔

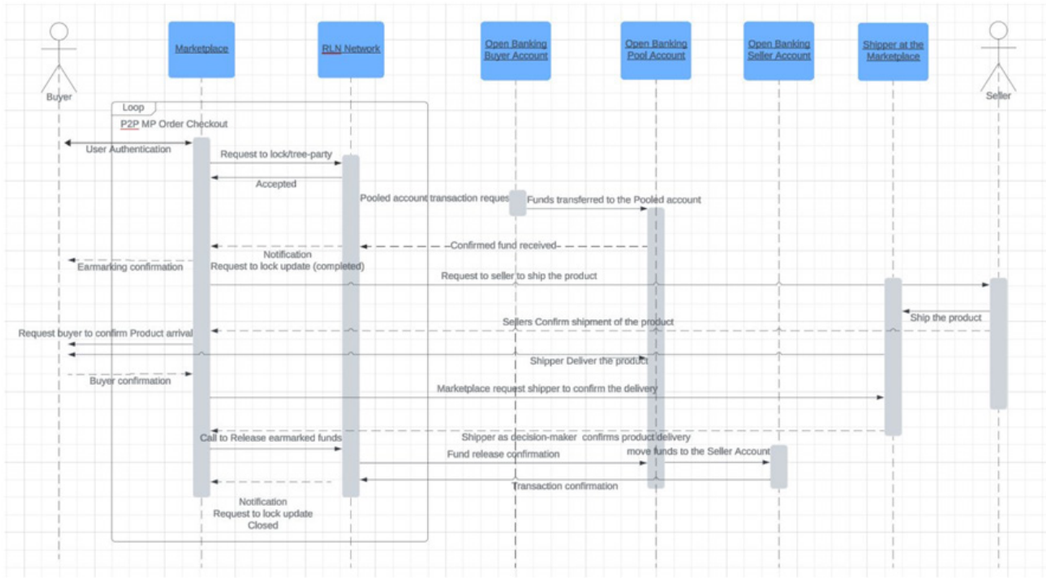
Technical Focus Areas by Use-Case

A full description of the Business Applications is beyond the scope of this report, but in what follows we summarise the key technical questions they were intended to explore or address.

1. Peer-to-peer ("P2P") marketplace

The key concept behind the first Business Application is the idea of 'cash on delivery', in which a customer 'locks' their funds before a product is shipped and the merchant is able to take ownership of the money as and when they can prove the goods have been delivered, irrespective of the form of money used by either party. The new set of interactions between consumers, marketplaces and sellers envisaged by this concept is depicted in Figure 5, below.

Figure 5. The Peer-to-peer marketplace Business Application explored the opportunity to reimagine the way in which risk is managed in online market places, through the use of locking capabilities.



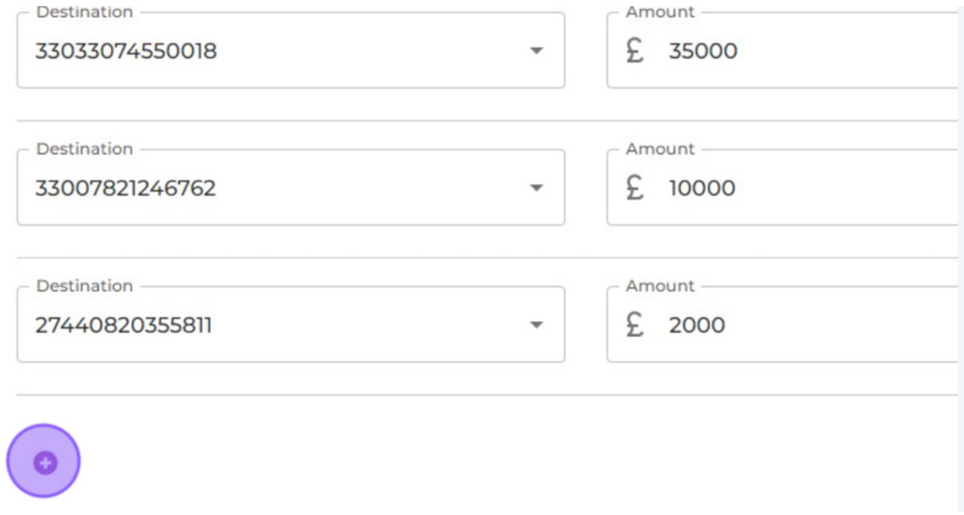
The key technical questions of interest therefore related to how a consistent 'locking' construct could be implemented across forms of money and across ledger types, as well as demonstrating the subsequent interoperability of money between these forms, and doing so in a way that provided a simple developer experience.

To achieve this, a simulated P2P marketplace application was designed and developed, and a significant amount of test data and interoperability flows defined and implemented, including support for the three models of tokenisation being studied by the project.

2. Home buying

The second Business Application used the risk and complexity associated with the completion of property purchases in England and Wales as its inspiration. Here, the key technical questions related to the implementation of multi-party locks, the synchronised execution of multiple payments, in a mixture of central and commercial bank money, and the monitoring thereof. Figure 6, below, shows an example screen developed for the scenario, where a set of disbursements can be configured, such that multiple payments are made at once when a purchase completes. Importantly, the sources and destinations of funds (not shown) can be any form of money and any type of ledger, demonstrating interoperability.

Figure 6. One of the concepts explored by the Home buying Business Application was the possibility of coordinating a multi-recipient set of payments, triggered upon the completion of a house purchase, irrespective of the form of money preferred by each recipient, or the ledger on which they choose to transact.



This Business Application addressed the need to demonstrate that each party could make its own choices as to the form of money they used and the type of ledger on which it was recorded.

3. E-commerce merchant gateway

The key technical question underpinning the third and fourth business applications was the extent to which the proposed UK RLN vision is, or could be made to be, compatible with existing e-commerce payment flows (this Business Application) and existing in-person PoS flows (Business Application 4).

The e-commerce scenario explored how the Experimentation Platform could allow commercial bank account details, simulated Digital Pound wallet aliases or tokenised deposit wallet addresses to be used by consumers to make payments online. Initiation of 'pull payments' across the infrastructure was demonstrated, and in a way that worked consistently across all forms of money.

4. Card integration and PoS compatibility

The card integration Business Application imagined a world in which retail CBDC and/or tokenised deposit platforms had been rolled out and cards had been issued to consumers that were backed by funds in these forms of money and type of ledger. The scenario considered the extent to which the existing payment card architecture, including its standard messaging formats and market structure, could accommodate these new forms of money. The potential latency of the 'authorisation' step was considered, given consumers' expectations around the near-instantaneous confirmation of card payments.

5. Settlement of tokenised bond

The final Business Application added a multi-asset wholesale perspective. An external bond tokenisation platform was simulated, and the use case explored settlement of bond issuances on a 'delivery versus payment' basis using the Experimentation Platform as a 'payment rail', irrespective of the form of money being used for payment.

For this use case, the focus was on a model where the tokenised bond platform was distinct from the Tokenisation Platform itself. So questions of interoperability, locking and atomicity were brought to the forefront.

For example, one scenario used a three-party hash timelock function provided through the API to lock tokenised deposits and central bank money in order to settle an issuance transaction.

A potential area of exploration for a future phase is a scenario in which the bond is issued to the *same* tokenisation platform as the cash (tokenised deposits and tokenised CBDC), with the differences compared.

Simulated Platforms

To support the development and testing of the Business Applications and of the Orchestration Layer more generally, a set of simulated underlying platforms were implemented. Although they are in some ways secondary to the key objectives of the project, they nevertheless represented significant pieces of work and the integration effort required yielded valuable insights for the project.

The key platforms that the project needed to implement were:

- A simulated Real-Time Gross Settlement (RTGS) platform, analogous to CHAPS, to enable testing of scenarios where settlement occurred over existing rails.
- An implementation of the BIS Project Rosalind API and back-end ledger, to enable testing of integration with a simulation of the Bank of England 'Platform Model' of Retail CBDC.
- An implementation of the Open Banking APIs providing access to several simulated banks, to enable testing of integration with existing comercial bank accounts.
- A simulated bank platform, to simulate scenarios where manual approval steps might be required.

Test Data

To support the above, a large set of test data was prepared. For each of the business applications, as well as to support anticipated testing of the API, a key requirement was that it be possible to demonstrate functional consistency across all forms of money and all ledger types. To support this, it was necessary to have 'accounts' and other data prepared at all simulated institutions.

Non-Functional Requirements

The key non-functional requirements to be demonstrated by the Experimentation Platform were data and transaction privacy, data security, API performance, and a mechanism to ensure controlled access to the Tokenisation Platform. The Solution Design chapter of this report below describes how these were implemented.

Other non-functional requirements including high availability, resiliency, observability, maintainability and throughput were out of scope for the implementation, but later sections of this report discuss how they could or should be implemented or explored in subsequent phases.

Limitations to Scope

Necessarily, within the constraints of a time-bounded project, many items had to be considered out of scope. The most noteworthy such items were as follows:

- As an Experimentation Platform, the solution design was intended to answer as many questions as possible. A practical consequence of this was that the focus was on proving what could be done (the 'happy path') rather than handling the edge cases and error scenarios when the answer was 'no'.
- Authentication between layers of the architecture was implemented using relatively simple approaches for the Experimentation Phase, but would need to be developed more completely for a production implementation
- The privacy design adopted an approach whereby a portion of the data submitted into the API could be encrypted so that counterparts could read it but the operator of the platform could not. The actual encryption was not implemented in the project, but the separation was provided and analysed.

04

Experimental Outcomes

Experimental Outcomes

In this chapter we summarise the outcomes of the project. We summarise feedback from third parties who developed Business Applications that used the API, as well as insights derived from building the platform and integrating it with underlying systems. We then explore three specific topics in more depth: functional consistency, insights for other projects, and the value of a shared ledger.

Key Outcomes

The technology workstream delivered a multi-issuer DLT-based Tokenisation Platform, and an API and Orchestration Layer. The key outcomes from this workstream were:

Foundational Capabilities

The project implemented a set of fourteen 'Foundational Capabilities'[25], such as push payments and interoperability across all forms of money. Five 'Business Applications' that used the Experimentation Platform to achieve a wide array of operations utilising these capabilities were built and tested. We found that the concept of a common API and Orchestration Layer across all forms of money and all ledger types is technically feasible.

The project demonstrated that a single API layer can support a payment where a buyer pays in retail CBDC on Project Rosalind and the seller receives payment as a tokenised deposit at a commercial bank, and a range of other combinations. This supports the hypothesis that it is possible to deliver a single set of APIs that allows third party innovators to deliver new products that work across all forms of money and ledger.

Tokenisation

A multi-issuer programmable Tokenisation Platform into which different forms of money (Wholesale and Retail CBDC, and tokenised deposits) are issued into their own partitions by each issuer was deployed and tested. Issuance of tokenised deposits and tokenised CBDC was simulated, and a set of transactions performed, demonstrating programmability and the potential for atomic settlement. Delivery-versus-Payment was explored, and this was tested using three different accounting models for tokenisation (direct, indirect – pooled, and indirect – shadow).

25 The full set of Foundational Capabilities implemented in the Experimentation Phase is described on page 25

- **Direct:** tokenised deposits are represented solely on the Tokenisation Platform.
- **Indirect – pooled:** tokenised deposits are represented on the Tokenisation Platform, with the aggregate amount in circulation tracked on a bank's core systems
- **Indirect – shadow:** tokenised deposits are represented on the Tokenisation Platform, with each wallet's balance being tracked in a 'shadow' account on the bank's core systems.

Each model has different implications for the cost of integration and the reconciliation flows required as a result. See page 60 for a detailed analysis.

Orchestration

The API and Orchestration Layer demonstrated that functional consistency across all forms of money is achievable, including examples of upgrading the capabilities of existing forms to match novel features of other forms. For example, a rich set of payment scenarios was implemented, showing payments to and from different forms of money and across different ledger types in a way that was transparent to the users of the API. Programmability of commercial bank money and some forms of funds locking were demonstrated

Production Insights and Contribution to Industry Debate

The Tokenisation Platform design included consideration of situations where fully pooled governance cannot be assumed, contributing to the broader industry conversation around the development of multi-national, multi-asset, multi-issuer, interoperable programmable ledgers. In particular, the project showed that an FMI-driven orchestration model for cross-ledger coordination is a feasible approach in such cases.

Issuance of CBDC and use for Settlement

The project demonstrated multiple models for the issuance of retail CBDC, and future work could explore in more detail the potential for a retail CBDC to be issued to a common platform alongside commercial bank money.

Settlement in central bank money was demonstrated via RTGS and with tokenised wCBDC, and integration with Fnality was studied. The use of tokenised wCBDC to facilitate settlement of tokenised commercial bank deposit transactions was explored, and opens the possibility of a potential new 24/7 atomic settlement capability facilitated by through tokenised wCBDC.

Platform for Innovation

The Experimentation Platform is a potential enabler of innovation, delivering – for example – interoperability, synchronised settlement, and tokenisation of commercial and central bank money.

Interoperability

Interoperability across various forms of money, including settlement in central bank money as well as across FPS, was shown to be possible and able to maintain the singleness of money and the role of wholesale central bank money as an anchor for trust and confidence.

Extensibility

The ease of development with the orchestration and tokenisation layers suggests each could indeed be a viable way to enable existing Foundational Capabilities to be enhanced and new ones to be deployed in an incremental fashion, thus enabling extensibility in a lower cost and controlled manner. This could be validated in a future hackathon

Atomic Settlement

When using an orchestration layer to synchronise the confirmation of transactions across two or more ledgers, the effects will become visible at slightly different times. This is unavoidable when the underlying ledgers are disparate, and so is a factor that must be taken into account when drafting rulebooks, for example.

However, when all assets are on the same shared ledger then atomicity becomes a possibility, and this may be advantageous in some situations, particularly in wholesale use-cases, and if tokenised wCBDC is available, and suggests further work to evaluate the use of a shared ledger for such scenarios could be beneficial.

Testing Approach

Key aspects to the testing and evaluation included:

- **Design workshops during the design of the Experimentation Platform and Business Applications.** An iterative design process allowed many of the key architectural decisions to be surfaced, and these are summarised in the Appendix, page 76.
- **Feedback from the third-party teams building the Business Applications.** Feedback on the usability and capability of the draft API was received, and is discussed below.
- **Direct experience from those building the platform.** The nuances of how to integrate an orchestration layer with a tokenisation platform, as well as the interfacing options for existing ledger systems, were explored directly during the process of development, and are discussed below.
- **Formal demonstrations of end-to-end use cases to large business audiences.** An iterative design process, in which feedback was solicited on the utility and relevance of the Business Application use-cases, was used to make the applications as realistic as possible and to inform the platform architecture.
- **Formal testing by the sponsors of the project.** The firms sponsoring the project were given the opportunity to 'on-board' to the platform in order to perform their own testing.
- **Hackathon for the innovation community.** Looking ahead, an open 'hackathon' for the innovation community is planned.

Business Applications: Feedback from Developers of Business Applications

One test of whether the Experimentation Platform has the potential to be a platform for innovation is the extent to which the third-party developers responsible for the suite of five Business Applications were able to use it. The feedback is that they were all able to achieve their goals, with all five Business Applications delivered on time and all of the Foundational Capabilities were exercised.

- **API stability and documentation.** The API and implementation were developed in parallel with the Business Applications being built to test them. The third-party developers stressed the need for stable and comprehensive API documentation.
- **Implications of asynchronous APIs.** The API relied heavily on asynchronous APIs. The model adopted eliminated the need for application developers to support inbound callbacks, but they were required to maintain state about outstanding asynchronous requests. A model that supports stateless client applications, that only want to repeatedly fetch all the notifications that are relevant to them, would be a valuable addition.

- **Helper functions.** One request for consideration in future phases was for an API endpoint allowing developers to get all relevant information relating to a particular lock or multilock without that information expiring. This was an example of a problem that could be solved within application code, but where a 'helper function' in the API could make developers' lives easier and hence make the platform more attractive. For example, a live implementation may want to add a "GET" API endpoint for retrieving all messages matching the client application's criteria.
- **Support for diverse programming models.** Another lesson for future phases is that rapid adoption of the API is probably most likely when it is possible for existing, and not just new, applications to connect. This may mean making the same functionality (such as lock queries) available in a number of different ways, in order to match the different programming models and assumptions used by the various applications. In other words, the more that the API can 'mould' itself to existing applications rather than require those applications to modify themselves to suit the API, the more likely it is to see adoption.

Learnings from Integrations with Underlying Systems

Orchestration Layer Privacy Design

The design explored in the Experimentation Phase was based on data intended for another specific institution being encrypted under their public key so that the platform operator is not exposed to it unnecessarily.

This approach to improving privacy versus existing models had previously been trialled as part of Project Rosalind. However, when implemented for the more ambitious UK RLN requirement, some limitations were discovered.

The orchestration layer is sometimes required to construct well formed messages to downstream systems (Open Banking, RTGS, Rosalind) that turn out to need data that is not available in the unencrypted portion of the messages. A subsequent phase should, therefore, perform further analysis of the data required by which downstream systems, and options for how it is obtained.

Should a modified version of the privacy model used in the Experimentation Platform be considered for future projects, the need for an associated PKI that all participants are obliged to use should be incorporated in planning and budgeting assumptions.

Fnality Service Connectivity

During the Experimentation Phase, we discovered that the Fnality API as it exists today is not compatible with the model used by the API and Orchestration Layer. This layer sends requests into the underlying ledgers as "Requests to Pay". This functionality is not currently supported by the Fnality API but could potentially be added to it. This prevented technical demonstrations of settlement via the Fnality method.

FPS Direct Connectivity

During the Experimentation Phase, the Orchestration Layer was connected to a range of simulated systems (Open Banking, RTGS, Project Rosalind, and the Tokenisation Platform), but connection to a simulator of FPS was not within the project's scope.

However, connectivity to FPS may be desirable for simplifying and improving consumer experience, enabling faster transactions and, possibly, simplifying bank integration. For this reason, some initial design work was performed, which is described later in this report. Should FPS integration become a requirement in the future, consideration would need to be given to obtaining appropriate authorisation for the platform to initiate transactions on FPS. The end-to-end data flow would need to be analysed to establish if the ISO8583-based messaging interface of FPS and a potentially richer format used by the Experimentation Platform does not result in the 'truncation' of any rich data or create a need for it to be conveyed over another channel[26].

Support for Point-of-Sale and E-Commerce transactions

The project explored the extent to which interoperability between forms of money could be exposed across the retail domain. For example, if a consumer has a card backed by a tokenised deposit, and wishes to use it to pay a merchant who prefers to hold CBDC, what changes to the existing infrastructure might be required.

Cards Support

Support for debit cards in UK point-of-sale (PoS) terminals was designed and demonstrated. A model based on the 'single message protocol' standard was utilised, which did not require any changes to PoS terminals for the Experimentation Phase scenarios. Routing via 'BIN range' was utilised, requiring configuration by card issuing systems, and associated back-end API integration into the platform. In this way, the experimental phase was able to use a model in which most payment card industry flows were unchanged, with integration performed by card issuer(s), demonstrating that merchants need not make changes to their existing terminals or processes. It may be the case that an automated update would need to pushed to the terminals from the card networks by the acquirers, and further investigation is required to clarify the exact process, but we understand that the merchants themselves should not need to take any action.

Payment Gateway Support

Support for direct integration of payment gateways with the API was demonstrated to in support of the third Business Application. The nature of the integration explored was a "Request to Pay" API call, with a webhook listener for the Experimentation Platform to deliver the response.

Integration of Tokenisation Platform into Bank Infrastructure

Three models for how to integrate a Tokenisation Platform into a bank's existing infrastructure were explored – 'direct', 'indirect – pooled' and 'indirect – shadow'[27].

They differ primarily in terms of how many existing bank systems and processes would need to be reconciled or integrated with to adopt the Tokenisation Platform.

We found that all models were feasible, but with different tradeoffs. For example, the 'shadow' model, where all updates on the Tokenisation Platform are reflected in near-real time back to tracking 'shadow' accounts set up on the bank's existing banking system might have lower impact on other bank systems, but it places a requirement that the accounts be kept in sync.

More precise quantification of the benefits of, and relative costs of integration for, the various tokenisation models could be an area of future work.

26 See page 53 for more detail on the design for FPS integration.

27 See page 60 for a detailed analysis.

Learnings about Functional Consistency

Functional consistency is the principle that different forms of money have the same operational characteristics. For example, if locking is to be made available on an rCBDC then a reasonable equivalent should be made available on commercial bank money. In this section we briefly summarise some of these important operational characteristics and the extent to which the Experimentation Platform demonstrated they could be made consistent across forms of money.

Acceptability: The API allowed funds to be transferred to and from all forms of money and between ledger types, and a wide range of combinations was tested by the Business Applications. This demonstrated that different forms of money and ledger could be used consistently.

Settlement Time: It was demonstrated that the Orchestration Layer could effect settlement in a range of ways: via RTGS and via tokenised wCBDC most notably. wCBDC, should it be made available on a Tokenisation Platform, has the potential to facilitate 24/7 settlement. This means, provided tokenised wCBDC could be used as a settlement mechanism for any type of transaction, the possibility of 24/7 settlement would be available for all forms of payment[28].

Payment Programmability: Programmability in the Orchestration Layer would be available for all forms of money, and the project demonstrated the deployment of several examples.

However, should users of a future Tokenisation Platform take advantage of its native programming model to extend the capabilities of any tokenised deposits, equivalent functionality may need to be deployed to the Orchestration Layer to ensure functional consistency for the other forms of supported money[29].

Interoperability between Forms of Money: The project demonstrated a large number of interoperability scenarios. Accounts were created for simulated payers and payers in all underlying ledgers, and using all forms of money. Payment flows were then tested for a range of combinations of ledger type and form of money for source and destination, and different settlement mechanisms (eg RTGS, wCBDC) were utilised.

Payment Infrastructure Compatibility: The Card integration and E-commerce Business Applications demonstrated functionally that consumers could pay in physical retail and online settings using any of the supported forms of money or ledger type, and that merchants could receive payments into their preferred form of money or ledger type. However, further work should assess the extent to which the non-functional requirements (such as latency in the cards domain) could be achieved.

28 Changes may be required in existing systems – e.g. to route settlements through tokenised wCBDC – to achieve this for those systems.

29 However, note that 'Functional Consistency' does not imply being completely identical in all regards. See https://arxiv.org/pdf/2308.08362 for a fuller discussion of the concept.

Learnings for related industry initiatives

This project is a contributor to the broader international debate around the desirability and feasibility of multi-national, multi-asset 'unified' ledgers or globally available 'layer one' networks. These initiatives share a similar thesis to RLN: that a multi-asset, multi-issuer, programmable tokenisation platform could deliver transformational value. And they observe that the path to such an outcome is likely to require the federation and/or merging of a collection of jurisdiction-specific and asset- or issuer-specific tokenisation platforms over time.

To date, however, no detailed technical concept has been developed for how such a vision could be achieved in situations where construction of a single shared ledger is not possible. The project implemented an FMI-driven orchestration model for cross-ledger coordination, in which the underlying ledgers implement a common protocol and can therefore share smart contract and other common capabilities, and explored this in the context of a wide range of retail and wholesale payments and asset tokenisation scenarios.

Demonstration of Shared Ledger Value Proposition

The architecture implemented for the Tokenisation Platform in the Experimentation Phase was a federated network of networks rather than a single fully shared ledger. At a functional level, this model was able to deliver all project goals: banks could issue tokenised deposits, they could be used to make payments, and could be settled in tokenised CBDC or over RTGS. Moreover, these tokenised deposits could be locked, and other programmability features such as the ability to impose constraints on how tokens could be spent were explored at the design level.

However, the project identified three areas where the specific design – single shared ledger or federated network of networks – had an impact on potential business outcomes. We summarise here findings on when an approach involving greater sharing could be preferable.

Atomic Settlement

When settling transactions spanning multiple ledgers, coordination of the process is required: the moments when updates to each ledger become visible at a technical level are likely to be different, if only by a very small amount of time. The design of the API and Orchestration Layer demonstrated how the necessary synchronisation process to coordinate these updates can be achieved across forms of money and across types of ledger.

However, in a *shared* ledger, with a common consensus process, a set of related transactions affecting several assets have the potential to be confirmed at a technical level in a manner that is atomic, and it may be possible to write rulebooks that reference this single technical event[30]. For example, if tokenised commercial bank deposits and tokenised wholesale CBDC were available on a single shared ledger, it might be possible to achieve settlement of a payment from a customer of one commercial bank to another, with no need for cross-ledger orchestration, opening up the possibility of 24/7 real-time atomic settlement.

30 Legal analysis is beyond the scope of this report, and this section limits itself to discussing the possibility of atomicity when using a shared ledger, and the possibility that a rulebook may choose to reference the moment of that event.

This model was not demonstrated in the Experimentation Phase. Instead, an alternative approach, in which each partition owner had full responsibility for confirmation of transactions in its partition was adopted, with locking and orchestration being used to synchronise settlement[31]. However, if tokenised wCBDC were to be available then the potential benefit of atomic settlement could be an argument in favour of the shared ledger approach, and a future phase could explore this[32].

Extensibility through Programmability and Composability of Business Logic

A key requirement of the project was to enable extensibility, such as the ability to add new foundational capabilities. The Experimentation Platform was therefore designed to allow extensibility at both the API and Orchestration, and Tokenisation Platform layers.

Programmability at the 'local' partition layer of the Tokenisation Platform was demonstrated. Asset locking is an example of this. In addition, 'global' programmability across partitions, such as that required to enable settlement of tokenised deposit transactions that cross two or more issuers, was demonstrated[33].

In the case of the Tokenisation Platform, an alternative programming model for extensibility becomes possible. The shared ledger architecture makes no distinction between how asset- or partition-specific logic and broader orchestration logic is deployed. Instead, all types of programmability can be implemented in a consistent manner. Such a model would be similar to how new logic is deployed to the public Ethereum network.

The simpler programming model afforded by this alternative approach could make it easier and faster for new capabilities to be deployed. However, this was not tested in the Experimentation Phase. The potential value of such an approach may come down to how often new logic would need to be deployed, and how much testing and review would be needed beforehand. A future phase could explore this hypothesis.

31 An alternative model, where assets on different ledgers are temporarily moved to a common location for the purposes of atomic settlement, was not explored in detail, but is discussed from page 89.

32 Note that in this report we focus on the technical 'reference points' – eg notaries – that the overarching rulebook and associated legal analysis may choose to refer to, and the technical implications of each. The question of when legal finality occurs under any given model is beyond the scope of this technical report.

33 As discussed earlier, the API and Orchestration Layer was reused to accelerate delivery, but the intended architecture is one where the Tokenisation Platform can coordinate its own cross-ledger transactions.

05

Solution Design

Solution Design

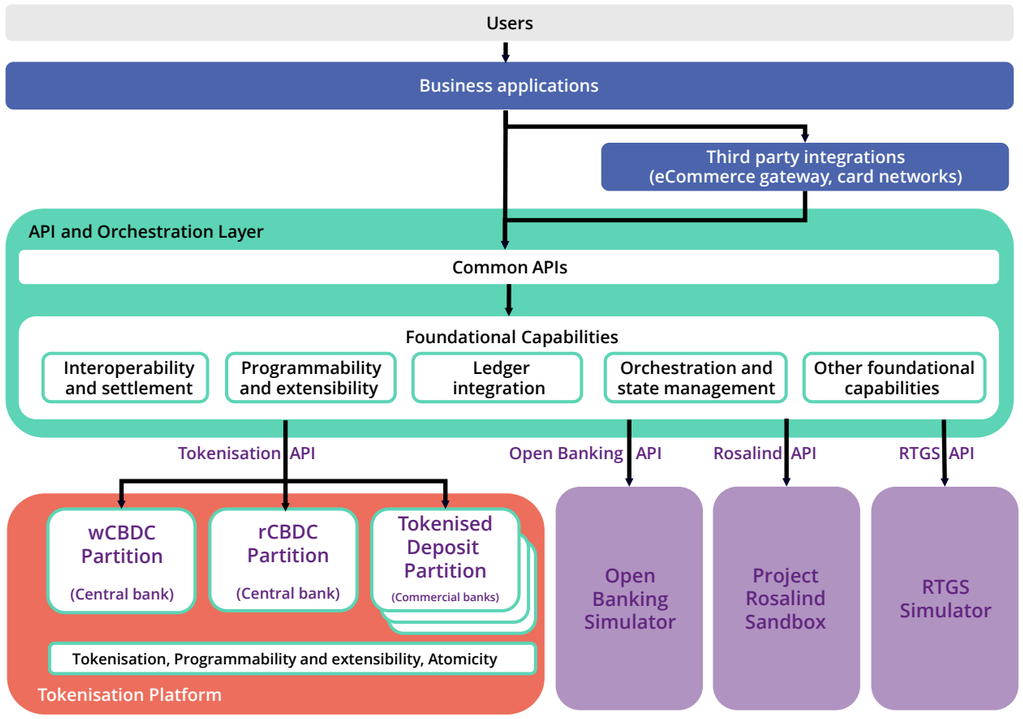
In this chapter we describe the high-level architecture of the API and Orchestration Layer, and Tokenisation Platform components, focusing on those that are most significant from an architectural or business decision-making perspective. We do not attempt to comprehensively document every aspect of the design.

This chapter is grouped into four sections. The first two sections relate to the project's primary deliverables. The second two sections are cross-cutting. The sections are:

1) The API and Orchestration Layer
2) The Tokenisation Platform
3) Programmability and Extensibility
4) Non-Functional Characteristics.

The overall solution architecture is depicted in Figure 7, below:

Figure 7. Experimentation Platform High Level Architecture



API and Orchestration Layer

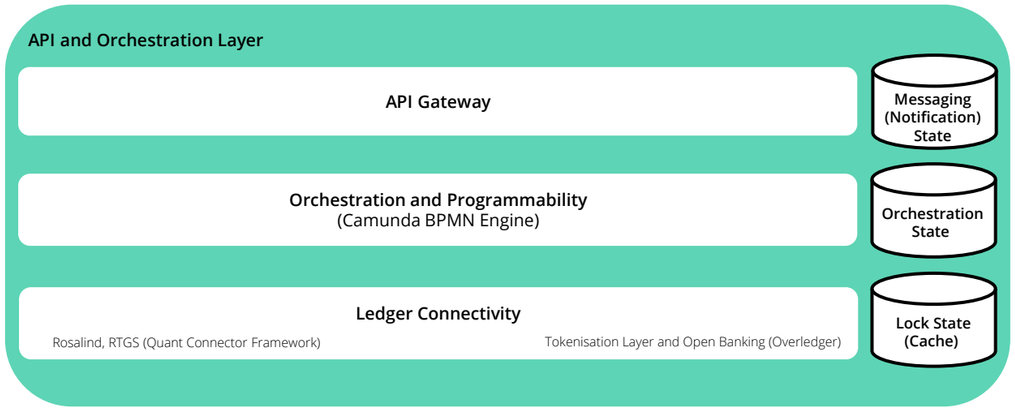
The API and Orchestration Layer serves as the central hub that interconnects all underlying systems. It unifies ledgers and assets, making them accessible to applications via a single API entry-point, and is the key enabler of functional consistency.

The key components are:

- The API Gateway, through which requests are received, and asynchronously delivered messages and other responses are stored awaiting collection
- The Orchestration and Programmability Layer
- The Ledger Connectivity Layer

Figure 8, below, illustrates the architecture of these components.

Figure 8. API and Orchestration Layer High Level Architecture



In this section we describe these key components, how they relate to each other and the broader solution, and explain how transaction privacy is achieved.

API Gateway

The purpose of the API Layer is to provide a single secure entry point that exposes the full set of APIs necessary to enable developers to experiment, innovate, and utilise various forms of money and ledger to build innovative new applications.

As such, the API abstracts a range of systems behind a unified interface, acting as a buffer between developers and the complexity and diversity of the underlying ledgers, programming styles and data models.

For the Experimentation Platform, the API Gateway exposed a REST interface, secured by OAuth2. As elaborated in the Architectural Decisions appendix below, this provided a consistent asynchronous programming model, with a notification service to collect responses[34]. In this way most developers only had to implement one invocation style irrespective of whether the operations they were invoking were low-latency/near-instantaneous or potentially very long-running.

34 There are a very small number of exceptions.

A consequence of an Architectural Decision to have no enrolment step (see page 70) was that there was no component that could map from generic identifiers to ledger-specific information. However, the API was nevertheless required to provide functional consistency across all forms of money, and to support the Experimentation Phase's objective of testing various ways of implementing certain capabilities.

The solution chosen was to expose the concept of 'Scheme Name' to developers through the API. This allowed developer to inform the system which ledger they wished to interact with, and to switch between different test cases (such as tokenisation models) on a case-by-case basis. This helped minimise the changes developers needed to make to interact with various parts of the system. Table 2, below, illustrates the specific schemes and the structures developed for this project.

Table 2. The concept of 'Scheme Name' was used to allow users of the API to declare which sections of the test matrix to activate on a case-by-case basis

Purpose	Scheme Name	Description	Example Identifier
Commercial Bank Money	UK.OBIE.SortCodeAccountNumber	First 6 digits are the sort code (maps to Member Id), the remaining digits are the account number	80200110203345
Commercial Bank Money with transitory tokenisation	UK.OBIE.T.SortCodeAccountNumber	First 6 digits are the sort code (maps to Member Id), (T.) the remaining digits are the account number	80200110203345
rCBDC (R)	UK.Rosalind.PIPIdAccountAlias	PIP ID before colon, alias, a hash of a telephone number or personal data after colon	ALKO00Z6NZYZEA6RLQ84:caab2ea35fe2d4aadad1173052403892657c6898dab89e59468956a4fa4d1c54
rCBDC (SL)	UK.RCBDC.RlnMemberIdAccountAlias	LEI before colon, alias, a hash of a telephone number or personal data after colon	ALKO00Z6NZYZEA6RLQ84:caab2ea35fe2d4aadad1173052403892657c6898dab89e59468956a4fa4d1c67
TD (SL)	UK.RLN.RlnMemberIdHashedSortCode AccountNumber	LEI before colon, hashed sort code and account number after	ALKO00Z6NZYZEA6RLQ84.I.caab2ea35fe2d4aadad1173052403892657c6898dab89e59468956a4fa4d1c67
Indirect TD (SL)	UK.RLN.I.RlnMemberIdHashedSortCode AccountNumber	Available for TD-TD transactions only. LEI before colon, letter I, hashed sort code and account number after. – this is used to indicate the settlement will be done via Open banking between backing accounts rather than wCBDC or RTGS.	ALKO00Z6NZYZEA6RLQ84.I.caab2ea35fe2d4aadad1173052403892657c6898dab89e59468956a4fa4d155

These schemes inform the Orchestration service about which underlying systems need coordination for conversions, transfers, and settlements to occur. This means that account identifiers passed into the API are necessarily (and deliberately) direct mappings of the actual account identifier of the underlying account being referenced.

In a production implementation it may be desirable to revisit the question of how to manage mappings of this sort.

Orchestration and Programmability Layer

The purpose of the Orchestration and Programmability Layer is to convert requests (such as 'transfer value from A to B') arriving via the API Layer into a request or, often, a sequence of requests to one or more underlying ledgers, each accessed through the Ledger Connectivity Layer.

In some cases, the underlying ledgers provided implementations of functions exposed by the API Layer, in which case the primary work of this layer was data transformation and routing. However, in many more cases, a complex sequence of activities is required, any of which might call out to an underlying ledger and which, as a consequence, may take some time to complete.

Due to the amount of variety in the experiments being supported – in the region of sixty flows, it became clear that workflow automation would accelerate the project[35]. The criteria for the choice of workflow engine included:

- Visual modelling front-end supporting BPMN format diagrams as used in the rest of the project
- Easy integration with Spring Boot to aid delivery timelines
- Extensive integration capabilities, preferably event-driven, to aid extensibility
- Open source, with a large active community

The platform selected for this component was Camunda[36], based on the strength of its workflow engine, its support for visual BPMN diagrams using a dedicated tool, ability to be embedded within the broader Java-based orchestration layer, and its use of a relational database to store flow data. The ability to embed the workflow engine within a larger Java program enables programming model choices such as the capabilities of the process engine to participate in Spring Managed or JTA transactions, and the threading model.

The architecture implemented a layered model, where the orchestration layer was abstracted from the specifics of how the underlying ledgers worked, which included not needing to know about their data models, interaction styles, security models and so forth.

The component responsible for mapping between the Orchestration Layer's more generic model and the specifics of the underlying ledgers was the Ledger Connectivity Layer, which is discussed in the next section.

It is hoped that the foundation of API and Orchestration architecture in the experiment can be built upon as a path to production. Most of the components of the experiment architecture can be reused, as each component has been designed as an indepdent microservice. Each microservice can scale horizontally and vertically individually and collectively across different environments, such as a highly-available critical infrastructure expected from a payment system. Further functionality for auto-recovery and cross-environment scaling between microservices could be designed and implemented in future designs.

35 In a live system with more statically defined flows, a workflow engine could still nevertheless be useful as an enabler of extensibility and programmability.

36 https://camunda.com/

There is further optionality to manage resilience by further decentralising the orchestration layer to be implemented within each participant's perimeter to share workloads and flows to have higher availability and throughput capability as opposed to a single centralised shared service.

State Management

The orchestration layer maintains a variety of information about in-flight transactions, requests and the statuses of locks and multi-locks. This state is held in a document-oriented database (Mongo Atlas). This choice was made based on its familiarity to the development team, its maturity, and its well-understood performance, availability and recovery characteristics.

Database components that handle the state of transactions, flows and programmability within the orchestration layer have been implemented as a single instance in the experiment, which provides a single point of failure. This is acceptable in The Experimentation Platform, but in production these critical components need 100% uptime and availability with realtime synchronisation to maintain state and integrity of data.

Ledger Connectivity Layer

Wherever possible, the Orchestration Layer was designed in a generic manner, with ledger-specific logic and integration being handled by the Ledger Connectivity Layer. A separate connector was deployed for each underlying system – such as Rosalind, the Tokenisation Plaform, simulated RTGS, and the simulated Open Banking system. These connectors share a common framework to simplify connectivity and integration.

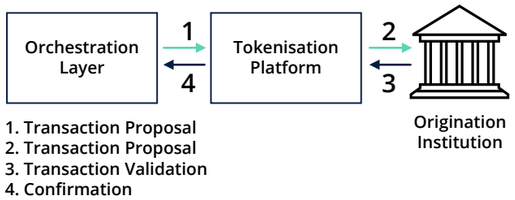
Connectivity to the Tokenisation Platform

Integration between an orchestration layer and a tokenisation platform poses particular challenges. While the tokenisation platform can be modelled theoretically as a a 'black box', behind a single endpoint, the reality is that it is a complex, distributed system in its own right, managing transactions among a group of distinct legal entities.

In the general case, a tokenisation platform that hosts partitions and nodes for multiple simulated institutions would present multiple endpoints at a variety of locations and the Ledger Connectivity Layer would need to route and connect to the appropriate target for each request and the solution was built with this in mind.

However, for the Experimentation Phase, the single endpoint model was adopted for simplicity. This is depicted in Figure 9, below.

Figure 9. Tokenisation Platform Connectivity



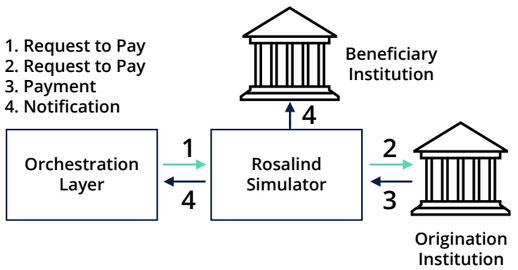
The component used to connect to the underlying Corda-based Tokenisation Platform was Quant Overledger. This is a pre-existing component that already has a series of orchestration flows modelled for this purpose and so it was adopted to help accelerate delivery. It did mean there was some conceptual overlap between the role of the Business Process Model and Notation (BPMN)-based orchestration layer and the flow features in Overledger, but this proved not to be an issue in practice.

Connectivity to the Corda-based Tokenisation Platform was implemented for the Experimentation Platform using Basic Authentication with username and password credentials, along with IP whitelisting. Callbacks from the Tokenisation Platform are validated using Overledger's OAuth 2.0 mechanism. The Tokenisation Platform has its own client credentials.

BIS Project Rosalind Connectivity

The Orchestration Layer communicates with the Project Rosalind API through the API Key authentication type. In addition, the Project Rosalind API also requires certian FAPI headers. These headers are x-api-key, x-idempotency-key, x-fapi-financial-id and x-jws-signature. The Rosalind API is broadly similar to that implemented for the overall Experimentation Platform API: mainly asynchronous REST invocation, with a similar notification framework. A typical request flow is depicted in Figure 10, below.

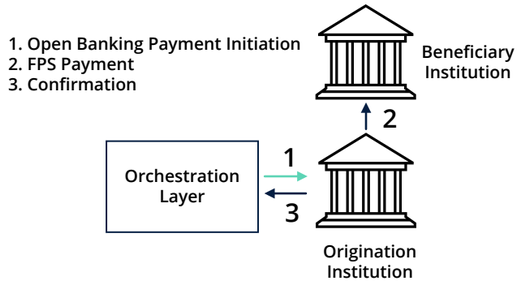
Figure 10. Project Rosalind Connectivity



Open Banking Connectivity

Overledger was used to communicate with the Open Banking simulator provided by Banfico, exploiting Overledger's support for the Open Banking FAPI specification[37]. Open Banking APIs were again primarly accessed via asynchronous REST invocation. A typical interaction is depicted in Figure 11, below.

Figure 11. Open Banking Connectivity



Central Bank Money Settlement

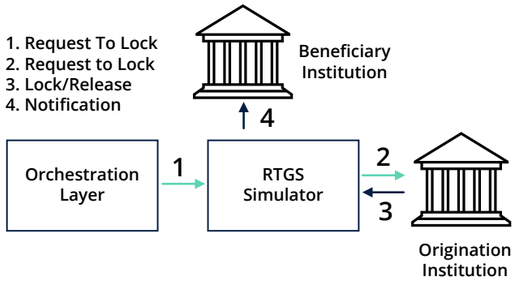
Three models for the provision of central bank money for settlement were explored: integration with RTGS, integration with Fnality, and the issuance of tokenised wCBDC onto the Tokenisation Platform. The latter is discussed in the Tokenisation Platform section, on page 88. The models requiring specific connector integrations are discussed here.

RTGS Connectivity

The Ledger Connectivity Layer communicates with the RTGS simulator through the Bearer Token auth type. The RTGS API was created for the project by a DXC team with prior RTGS implementation experience, and the synchronisation model was based on public materials available from the Bank of England. The RTGS API is a synchronous REST API. This is depicted in Figure 12, below.

37 https://openid.net/wg/fapi/specifications/

Figure 12. RTGS Simulator Connectivity

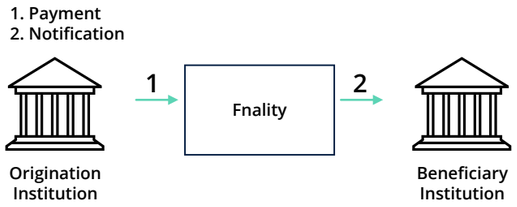


Fnality Service Connectivity

Connection to Fnality would potentially allow the orchestration layer to perform settlement transactions on a 24x7 basis, meeting some of the needs for immediate settlement. Unlike wCBDC issued directly on the tokenisation platform with shared consensus this would not allow for atomic settlement within that layer, and would instead still require synchronisation coordinated by the orchestration layer. However, it does open the possibility that Direct Tokenised Deposits could be used for immediate payments with immediate settlements.

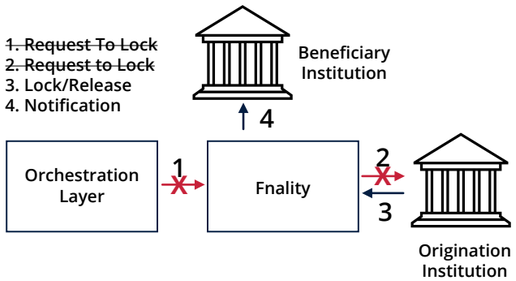
Unfortunately, we were unable to implement connectivity with the Fnality service, as the current API does not support 'payment request' flows. The model explored in the Experimentation Phase, as described in the above sections, does not directly instruct back-end ledgers. Instead, it asks them to request that financial institutions take action to instruct the ledgers. The Fnality service today has an API that supports the following model instead, depicted in Figure 13.

Figure 13. Actual Fnality Service Flow



For the model used in the Experimentation Platform, payment request messages are needed. The flows that would be needed are shown in Figure 14, below:

Figure 14 Fnality Service required flows



FPS Connectivity

FPS Connectivity was not demonstrated in the Experimentation Phase but consideration was given to how it might be achieved, which we discuss here.

The Faster Payments System (FPS) is a retail payment system in the UK used to transfer funds from one account holding institution to another. Detail of FPS and its participation models are available from Pay.uk[38]

38 https://www.wearepay.uk/what-we-do/payment-systems/faster-payment-system/

Direct connectivity is subject to scheme rules, which could pose an obstacle for a centralised version of the platform. However, a distributed version might be able to use existing scheme memberships held by institutions, subject to confirmation with Pay.uk and this could be a topic for future work.

It may be possible to configure the Orchestration Layer as an 'indirectly connected' participant. However, this may restrict the functionality available to FPS members that support real-time 'indirect' FPS interfaces, and performance may be variable depending on the institution. The rest of this discussion focuses on direct connectivity.

We assume that the Ledger Connectivity Layer would make use of an existing FPS-Certified Gateway. These provide simpler and more modern interfaces, and allow the use of ISO20022 messages. FPS will not necessarily accept all such data today[39] but future versions of FPS or the New Payments Architecture (NPA) may do so.

To act on behalf of members as an FMI or TSP, the orchestration layer would need to obtain authorisation from account owning institutions before initiating transactions on FPS from accounts owned by that institution. This is the primary difficulty with FPS integration: all connected participants either own customer bank accounts or act on behalf of (and with the authorisation of) an institution that does.

The implications for the Future Work chapter, below, highlights the need for design work in this area.

Card Support

Support for debit cards in UK PoS terminals was designed and implemented. The outcome was as follows:

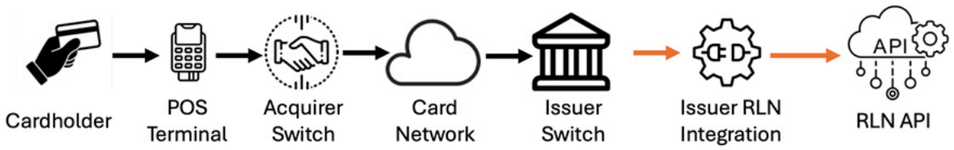
- While 99% of UK transactions use the 'dual message' standards, these are not appropriate for the Experimentation Platform design as money is being transferred at the time the payment is made. In a dual message system, no payment order has been issued and no money can be moved until the settlement file is received, often at end of day, but possibly up to eight days later.
- The single message standard does create a payment order at the time of the transaction happening and is compatible. The single message protocols are supported on UK PoS terminals and do not require any changes.
- After routing through the card network, the transaction can be recognised by BIN range/message type (this requires the card schemes to define the message type) and be routed appropriately. This requires configuration on the card issuing host system and appropriate back-end API integration with the Experimentation Platform API.

Support for cards in the project follows the pattern deployed for other types of non-standard assets backing cards. There are several examples of this type of card with, for example, gold, cryptocurrency or CBDC backing.

39 Further work should validate this; the statement above is intended only to observe that ISO20022 messages can hold more information than ISO8583 messages in general.

Most of the card flow and card settlement processes are unchanged. The integration occurs in the infrastructure of the card issuer, and transactions using tokenised deposit accounts are routed by the issuer's card switch, then transformed from ISO8583 into an API payload for the Experimentation Platform API. The API call debits the customer account and moves the tokens into a holding account (to be converted and settled at end of day), as shown below in Figure 15, below.

Figure 15. Card Support in the Experimentation Platform



In summary:

- The card issuing member does need to integrate the card flow with the platform.
- We believe merchants will not need to make any changes to their existing processes, but it is possible that updates to terminals may need to be pushed down by acquirers.

Payment Gateway Support

Support for the direct integration of payment gateways with the API was demonstrated in the third Business Application. The integration required was as follows:

- The payment gateway operator would need permission to access the API.
- The payment gateway is integrated with the API in a similar way to other API-based payment rails (such as PayPal, Open Banking, VISA Direct, or Mastercard Send).
- The nature of the integration is a "Request to Pay" API call, and a webhook listener for the API and Orchestration Layer to deliver the response to.
- The latency of responses will be determined by the consumer consent flow required. In the Experimentation Platform, this is an immediate and automatic response. However a manual consent may be needed in reality.

Locking

An exception to the strict layering discussed above was the implementation of locking capabilities. In particular, for ledgers without native support for locking, the most obvious approach would have been to simulate the feature at the orchestration layer, relying on the state database to keep track of locks. However, the Overledger component, already in use as the connector for two ledgers, had rich support for features such as locking, hash timelocks, and three-party locks out of the box. This was therefore used to accelerate delivery of these capabilities.

Security and Privacy

We implemented a principle of limited visibility, ensuring that the Experimentation Platform, and downstream systems, only accessed the data they needed to see, and the rest could be encrypted under the public key of the intended recipient institution. Implementation of this encryption was deliberately excluded from the Experimentation Platform implementation to enhance speed of delivery and ease of use during the Experimentation Phase and so should be validated in a future phase. However, the API was documented to indicate which data should be encrypted. This approach is described as an Architectural Decision, page 80.

As a result, the privacy model was designed with the intention that service user and customer data is protected and that data sharing between participating institutions occurs only when absolutely necessary for transaction purposes. Non-account-holding users of the API do not have access to service user data and can only issue "request to pay" or "request to lock" API requests, which require approval from the relevant account-owning institution.

We now outline some key privacy features incorporated into the design

Consumer Consent

The Experimentation Platform does not include a consent module for transactions occurring solely on the Tokenisation Platform or Project Rosalind. However, the OBIE specification consent module is incorporated into the Open Banking simulator to demonstrate what could be possible. This module assures that no transaction initiation or account balance retrieval on Open Banking-accessed ledgers occurs without the consumer's explicit consent. Furthermore, only account-holding participants can directly interact with service user data. Non-account-holding participants must request account-holding members to submit transactions or locks on their behalf.

Data Minimisation

Data is only stored in or processed by the orchestration layer when absolutely necessary. To ensure customer data privacy, access is restricted through client credentials of the member applications. For instance, only Bank A can use the GET API method to view transactions and locks it has created. Similarly, non-account-holding institutions can only view their own requests to lock or requests to pay.

There are three main categories of data persisted at different times by the system:

- Information relating to in-flight transactions, locks and caches of the state of underlying ledgers
- Identity and account mappings, linking various identifiers across different systems. Given that enrolment was not a requirement for this phase, this information is limited to service user information automatically tracked by the orchestration layer rather than information about individual users of the system.
- Asynchronous notifications being stored pending collection by API callers.

Summary

The API and Orchestration layer can be thought of, conceptually, as a simple three-layered architecture: an API entry-point, a BPMN-based workflow engine with state management, and a set of connectors. However, unique requirements such as support for locking, the nature of a tokenisation platform as a multi-entity distributed platform, and some subtle data privacy requirements mean that the design is more complex than may first be appreciated.

Tokenisation Platform Implementation

The project implemented a 'federated network of networks' model built on Corda for the Experimentation Phase. In this section we detail how this was implemented. The design decisions leading to this model are discussed as Architectural Decisions, page 71. Figure 16, below, depicts the overall architecture.

Figure 16. High Level Architecture of the Tokenisation Platform, showing the option to utilise a DLT platform's in-built cross-partition orchestration capabilities (although this was not implemented for the Experimental Phase)

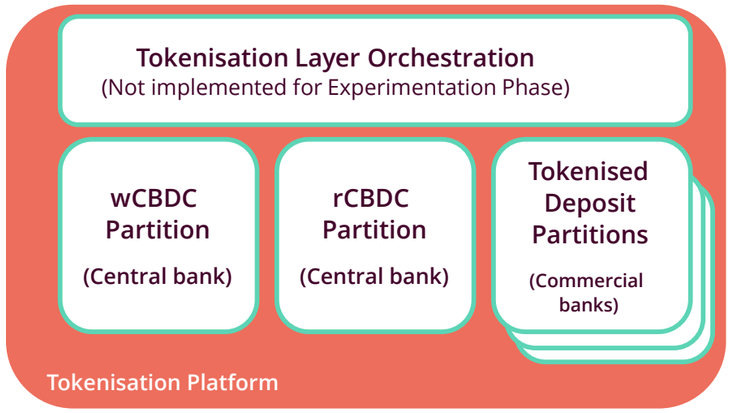
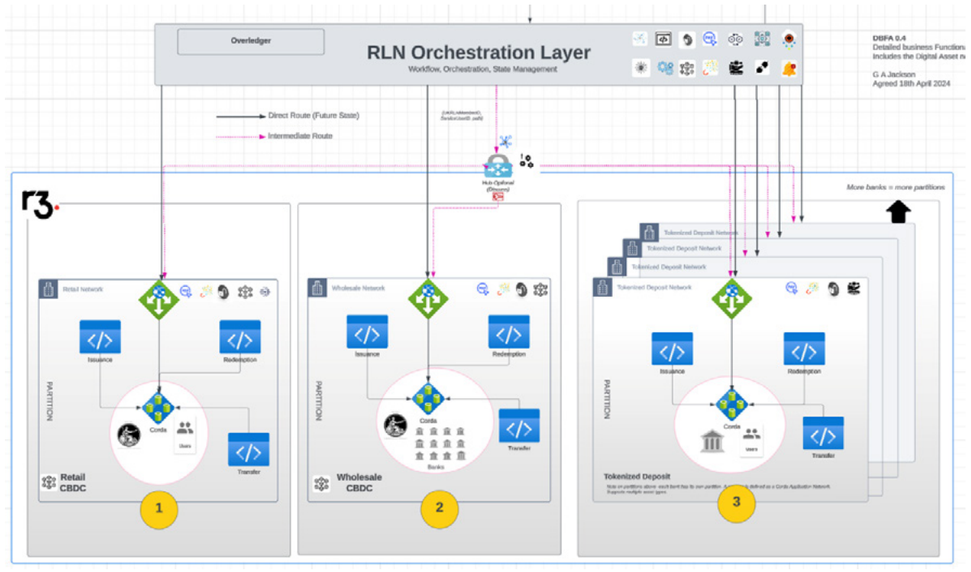


Figure 17, below, provides the next level of detail.

Figure 17. The Tokenisation Platform architecture explored in the experimentation phase consisted of a series of Corda application networks, one per partition, federated into an overall multi-cluster Corda deployment.



The box in Figure 17 labelled (1) is a partition, where the central bank is the owner and can issue/redeem/transfer rCBDC, and the other participants (commercial banks in this case) would handle retail users.

In (2) we see a second Corda Application Network as part of the same overall Corda deployment but in this case for wCBDC. Here, the other network participants are virtual nodes representing the commercial banks.

These networks offer the same 'issue', 'redeem' and 'transfer' functions.

In (3) we see a collection of Tokenised Deposit Application Networks, one per issuer, again offering the same functions and again part of the same overall distributed Corda deployment.

Importantly, any party that participates in or controls more than one partition can do so using the same Corda cluster, with this cluster managing their presence in each partition (their virtual node). Participants can deploy this cluster for themselves or obtain access from a service provider such as the operator of the platform. This means that each participant can utilise one Corda deployment that they use to participate in all partitions.

Corda Architecture Overview

In this architecture, the networks share some common features:

- Each application network is '*permissioned*', with a network operator deciding who can and cannot join such a network. In the Experimentation Phase, all clusters were operated centrally, but in a production deployment of this architecture, this permissioning model would allow each partition owner to assume responsibility for controlling access to their partition.
- Each member of each application network has a virtual node hosted in a Corda cluster. Each virtual node's identity is secured by a PKI controlled by the relevant application network operator[40].
- Each virtual node's ledger data is stored in a database.
- Each network has a 'notary' that confirms the proposed transactions in that application network, and which can be operated by or on behalf of the entity (e.g. bank) whose partition it is, thus providing a model where each issuer can be in full control of the movement of its assets.

Corda Clusters

The Experimentation Phase operates across five discrete clusters. This is intended to simulate a situation in which four banks decide to run their own infrastructure, and all the others rely on Corda infrastructure provided by a central operator. Conceptually, one can think of a cluster as being a physical Corda installation, controlled and operated by a single institution, where the most common deployment models are (1) a cluster operated by a specific institution to manage its interactions on all the Application Networks of which it is a part or (2) a cluster operated by a central operator, hosting virtual nodes for any and all participants who wish to utilize this shared service rather than run their own nodes[41].

A cluster is a deployment of Corda under the control of a single operator or administrator. It can host one or more entities, represented by virtual nodes for each application network of which they are a part, and manages a set of 'workers', which are processes that perform functions on the network on behalf

40 Identity management in Corda is explained in more detail in the product documentation: https://docs.r3.com/en/platform/corda/5.2/application-networks/managing.html

41 To optimise hardware costs in the Experimentation Phase, banks were evenly distributed across the five clusters rather than having one cluster per bank for four banks, and the rest on a single shared cluster as implied by the body text.

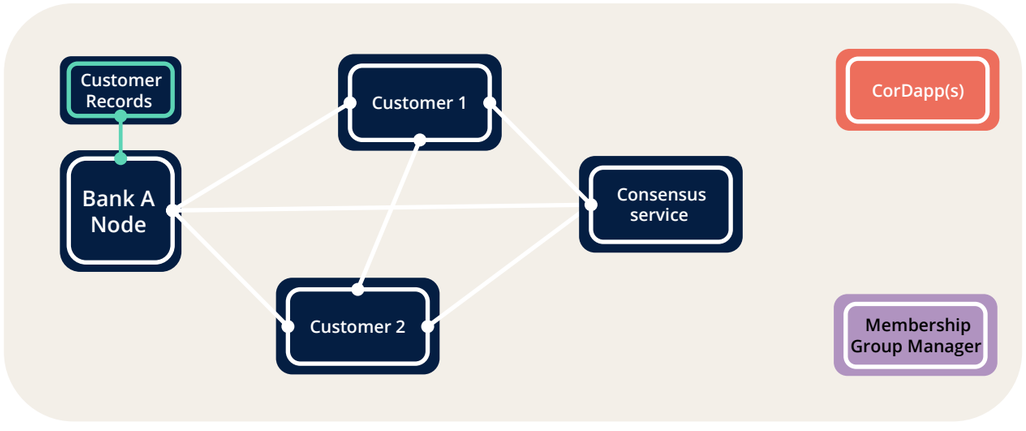
of the virtual nodes, such as processing transactions, executing workflows or interacting with the database.

In the case where a cluster only hosts workers for a single legal entity it is conceptually equivalent to a 'node' in other blockchain architectures. However, the ability to host workers for multiple entities means Corda clusters can provide multi-tenancy, mapping more naturally to the typical DLT deployments we see in the wild, where many participants outsource the operation of their 'node' to a third party whom they trust (e.g. to treat their data appropriately).

Application Networks and Partitions

The Experimentation Phase modelled each partition as an 'Application Network'. An Application Network is a set of virtual nodes configured and permissioned to transact with each other for a specific purpose. These nodes all have access to the same application (smart contract) logic, their permissions are controlled by a Membership Group Manager operated by or for the Application Network's operator (that is, the partition owner in this case), and they rely on a common notary. This is depicted in Figure 18, below. In the Experimentation Phase we utilised an Application Network for each tokenised deposit partition, where the members of any given network are the tokenised deposit issuer and its customers.

Figure 18. An Application Network is the combination of a specific smart contract application, a set of users (participants, represented by virtual nodes), a notary (consensus service) and a Membership Group Manager that controls access.



Importantly, an Application Network can span multiple Corda clusters. Specifically, a Corda cluster is the physical unit of deployment and operation of a Corda installation, and can host one or more identities, with a virtual node for each Application Network with which they interact.

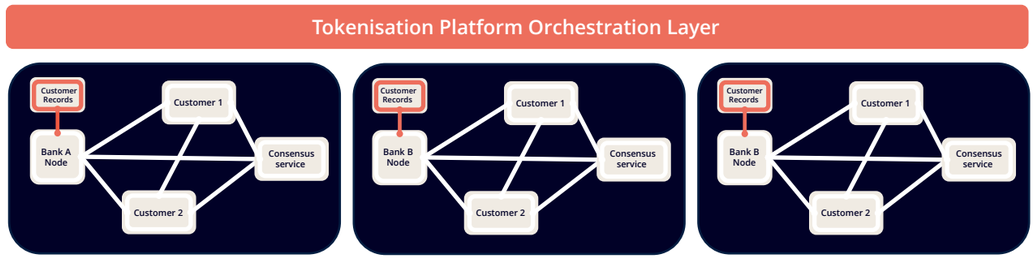
In summary, each participant, such as a bank, participates in the overall Corda deployment by running or consuming the service of a Corda cluster. This cluster, in turn, manages that party's participation in one or more Application Networks. The cluster acts as the participant's single entry and coordination point to/across all Application Networks, and each Corda cluster is typically implemented on a Kubernetes cluster, operated by a specific organisation.

Coordination across Application Networks

A deployment consisting of multiple Application Networks requires coordination when cross-partition operations are required.

The model explored in the Experimentation Phase is one in which assets never leave their 'home' networks and, instead, the underlying platform's coordination and locking capabilities are used to effect these settlement operations[42]. This is depicted conceptually in Figure 19, below, where assets issued by Bank A, say, are managed within the Bank A partition on the left.

Figure 19. In the Experimentation Phase, assets are always associated with their 'home' partition (Application Network), and cross-partition operations are coordinated by an orchestration layer



For tactical project management reasons, the design adopted for the Experimentation Phase differed in one important respect to that which would be recommended for a production environment. Specifically, it 'piggybacked' on the wider project-wide Orchestration Layer to meet some coordination requirements, and so the capabilities of the 'Tokenisation Platform Orchestration Layer' shown architecturally above were provided by the API and Orchestration Layer for the purposes of the project.

However, orchestration across tokenised deposit partitions could also be delivered using native workflow capabilities within a DLT platform. Indeed, for a production implementation of an Application Network architecture this would likely be the preferred approach. This is because any operations that only affect the Tokenisation Platform could be completely executed by that layer, without relying on or even assuming the existence of a broader orchestration capability.

Summary

The tokenisation platform implements a 'federated network of networks' model. It is deployed across several Kubernetes-based Corda clusters, delivering fourteen tokenised deposit partitions. The design enables tokenisation of commercial and central bank money and synchronised settlement across partitions, in a way that gives each partition owner a considerable degree of autonomy. The programmability and extensibility capabilities of the platform are discussed below, and the extent to which the design provided differentiated value is explored in the Experimental Outcomes chapter above.

42 This design decision is explored as a set of Architectural Decisions, page 78.

Support for Different Tokenisation Models

The fundamental purpose of the Tokenisation Platform is to facilitate the tokenisation of deposits and other customer assets. In effect, it facilitates the 'projection' of portions of a bank's balance sheet or custody systems onto an infrastructure that is operated in common with other institutions in the industry.

A comprehensive deployment of such a platform by a bank would entail integrating it with a large range of existing systems: accounting, customer servicing and statementing, transaction monitoring and so forth. This might enable the bank to fully exploit every capability of the new platform, but it could be costly. Therefore, approaches which require less integration could be attractive.

In this section we summarise three models for how a Tokenisation Platform could be integrated into bank systems – direct, indirect and shadow – as well as the key aspects of the design needed to support each one. All three approaches were explored in the Experimentation Phase.

The Direct Model

The Direct model can be thought of as the 'maximalist' approach. This model views the bank's partition in the Tokenisation Platform as a peer of the traditional ledger, requiring integration with the same systems as the existing ledger.

This is depicted graphically in Figure 20.

Figure 20. In the Direct model, consumer liabilities are represented in either the traditional ledger or the Tokenisation Platform, but not both. To calculate the bank's total consumer deposit liabilities, the total on both ledgers must be summed. Similarly, all other systems would need to integrate with the new ledger

Customer A	Customer B	Customer C	Customer D	Total Traditional Ledger Liabilities
100	150	75	50	375

Traditional Ledger (eg Core Banking System)

Total Consumer Deposit Liabilities = 375 + 225 = 600

Customer A	Customer B	Customer C	Customer D	Total Tokenisation Platform Liabilities
50	25	100	50	225

Tokenisation Platform

This means, for example, that any liabilities transferred from the traditional ledger to the Tokenisation Platform would leave the traditional ledger: they would be recorded in one or the other but not both. So, finance and accounting systems, as well as customer statementing and so forth would need to integrate with the new ledger.

In some cases a bank may choose to implement a 'mirror' account in the traditional ledger to 'track' what is happening in the Tokenisation Platform at an aggregate level and this may reduce the integration burden somewhat.

The Indirect Model

In the Indirect model, a new account – which we could call the 'pool account' – is created on the traditional ledger. When a customer wishes to transform a traditional deposit into a tokenised deposit, the requested amount is transferred to the pool account and an equivalent amount is issued onto the Tokenisation Platform.

In this way, we can think of the Tokenisation Platform as a sub-ledger of the traditional ledger: the entirety of the bank's liabilities to its retail customers is still captured by the traditional ledger, but to see how each individual tokenised deposit is allocated you must look at the Tokenisation Platform. This is depicted graphically in Figure 21.

Figure 21. The Indirect model imagines the Tokenisation Platform as a subledger of the traditional ledger. Here we see four customers – A to D – whose deposits are split across the two ledgers. For example, customer A has £150: £100 is recorded in the traditional system and £50 exists as tokenised deposits, recorded in the Tokenisation Platform. It is possible to calculate the bank's total consumer liabilities simply by summing across the accounts in the traditional ledger (including the 'pool' account). But to see a breakdown of individual consumers' tokenised deposits one must interrogate the Tokenisation Platform. This model still requires integration effort on behalf of the bank but it is mostly restricted to systems that require access to transaction-level information; systems operating on aggregate information can continue to feed from the existing system.

Customer A	Customer B	Customer C	Customer D	Pool Account	Total Consumer Deposit Liabilities
100	150	75	50	225	600

Traditional Ledger (eg Core Banking System)

Customer A	Customer B	Customer C	Customer D	Total Tokenisation Platform Liabilities
50	25	100	50	225

Tokenisation Platform

This approach might have limited implications for departments within banks that do not need the individual tokenised deposit balances or tokenised deposit transaction info, since they could obtain the aggregate tokenised deposit liability from the traditional ledger. However, it would have deeper implications for other departments. For example, any system that needed to know an individual's total deposits (across both systems) or needed access to the underlying transactions would need to integrate with both ledgers.

The Shadow Model

In this model, 'Shadow' accounts are created in the traditional ledger to track the activity of each wallet on the Tokenisation Platform. Any activity (such as a transfer between wallets) on the Tokenisation Platform is synchronised back – 'shadowed' – to the traditional ledger.

The implications for the bank's existing systems should be minimised under this model (although not zero[43]), since the traditional ledger could still be a source of data for other systems. However, the tradeoff is that the shadow accounts need to be maintained.

In the Experimentation Platform the orchestration layer was used to manage this process, using the Open Banking interface to generate the appropriate transactions. This meant that the existence of – and requirement to maintain – these accounts needed to be visible to the orchestration layer. And because 'enrolment' was out of scope (see page 78) it required the shadow account details to be provided via the API.

For a production implementation, it is likely that responsibility for maintaining these accounts would fall on the banks themselves. And in the event that the orchestration layer did take responsibility, an enrolment step would reduce the need for the details to leak through the API. It is possible that an alternative model based on reversing payment flows, such that beneficiary banks, who posses the required information, are responsible for triggering the payment, as a 'request to pay' may obviate the need for enrolment but this was not been fully explored in the project and is a potential area for future work, as is exploring whether this synchronisation activity should be the responsibility of the orchestration layer operator or the bank itself.

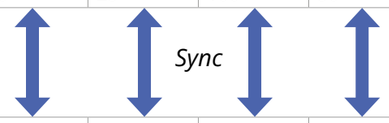
The Shadow model is depicted graphically in Figure 22.

43 For example, the capacity of the existing system to manage a large number of new shadow accounts would need to be assessed, and work performed to track the relationship between a customer's regular account and the shadow, etc.

Figure 22. In the 'Shadow' model, everything that happens on the Tokenisation Platform is synchronised back to 'shadow' accounts on the traditional ledger.

Customer A	Customer B	Customer C	Customer D	Customer A TD	Customer B TD	Customer C TD	Customer D TD	Total Consumer Deposit Liabilities
100	150	75	50	50	25	100	50	600

Traditional Ledger (eg Core Banking System)



Customer A	Customer B	Customer C	Customer D	Total Tokenisation Platform Liabilities
50	25	100	50	225

Tokenisation Platform

Programmability and Extensibility

A stated aim of the project is that two related concepts – programmability and extensibility – be explored. In particular, the design needed to demonstrate the money and other assets managed by the assets could be made *programmable*, and that the platform itself be *extensible*.

In this section we summarise which types of programmability were explored, and where in the architecture, before reviewing how the platform could, in principle, be extended by its operator or other parties.

Programmability

Three types of programmability were considered as part of the design:

- Programmable spending constraints.
- Event processing capabilities
- Complex workflows

Programmable spending constraints include locks, but can, in principle, encompass concepts such as 'purpose bound money'. Locks were implemented in both the Tokenisation Platform and in the API and Orchestration Layer.

Event processing capabilities were not explored in depth, but both the Tokenisation Platform (via the Corda Flow Framework) and the API and Orchestration Layer can be used to provide such capabilities when required.

Dozens of complex workflows were implemented in the API and Orchestration Layer, but this capability was also demonstrated at the Tokenisation Platform in the form of cross ledger synchronised settlements.

Extensibility in the API and Orchestration Layer

Extensibility in the API and Orchestration Layer potentially has a different scope to extensibility in the Tokenisation Platform. While the Tokenisation Platform can be extended by members for their needs in ways that do not affect the orchestration layer, part of the purpose of the orchestration layer is to provide functional consistency across all forms of money.

Extending the Tokenisation Platform can result in either differentiating functionality for individual patition owners, or collectively, such as support for non-consumer use cases. By contrast, extending the orchestration layer extends the capabilities of the system as a whole.

We have identified three major ways the orchestration layer could be extended, depending on the final design:

- Extending capability by surfacing new functionality from the Tokenisation Platform in the API
- Extending the range and complexity of the workflows in the orchestration layer via its workflow engine
- Implementing a "programmable payments" execution environment for complex logic, potentially in a domain specific language. This option could be the subject of further work.

As mentioned in this chapter above, the Corda layer of the Tokenisation Platform can be extended to implement new functionality, potentially operating as a test-bed for new capabilities. When capabilities are shown to be attractive it may then be possible to expose that functionality to the Orchestration Layer and, in turn, provide simulations where necessary above the other ledger types. This would require cooperation and coordination between the developer of the extension and the platform operator to deploy into a production environment.

Turning attention to the API and Orchestration layer itself, the workflow engine selected for the Experimentation Platform has been demonstrated to be flexible and straightforward. Therefore, it is possible that network participants or other parties could run development instances of the orchestration layer and develop new workflows. The packaging format used by the workflow engine allows workflows to be shared with the platform operator for analysis and testing, prior to trial deployment as part of a careful change control process. In this way, participants would be able to prototype and test new functionality locally and subject to agreement, see that functionality rolled out to the production system.

The final possibility is to provide a suitably secured and hardened programmable payments execution environment, to which any authorised member (or indeed service user if desirable) could write and deploy workflows and automations specific to their use cases or accounts. This could be an end-state goal on the journey towards a comprehensive platform for innovation. However, it comes with obvious risks. This option was not included in the Experimentation Platform but could be explored as further work following on from it.

Extensibility in the Tokenisation Platform

The key mechanisms through which the Tokenisation Platform can be extended are by:

- The addition of applications (sometimes referred to as smart contracts)
- Augmenting the capability of existing applications
- Deploying orchestration or composability logic that combines the capabilities of existing applications.

In the Corda architecture, this can be achieved by deploying new Corda Applications ('Cordapps') either into an existing Application Network or to a new one. Some of the key ways in which new or changed Cordapps can extend a solution or network are:

- New or enchanced contract logic, written in JVM languages such as Java or Kotlin.
- New or enhanced workflow logic, utilising the Corda Flow Framework.
- Deployment of new 'Corda Workers' into a Corda Kubernetes cluster, accessed over the internal Kafka bus (future).

Non-Functional Requirements

The Experimentation Phase was primarily focused on answering functional questions. However, some non-functional questions are inextricable from the UK RLN vision or the way in which a production system would be regulated. The key points are discussed in this section.

Data and Transaction Privacy

The system contemplated by this project could potentially have access to significant amounts of personal data. Some of this is unavoidable, and the operation and regulation of the platform would need to take this into account. However, two specific concepts were explored to reduce the amount of personal data shared within the system.

- The 'encrypted payload' concept in the API; and
- The design of the Tokenisation Platform

The encrypted payload concept is based on the idea of enabling an arbitrary ISO20022 message to be included in requests to the API, encrypted under the public key of another participant, utilising a PKI operated by the platform operator. It is core to the design of the API and so is discussed in more detail above.

The data and privacy design for the Tokenisation Platform rests of the following foundation:

- The standard Corda 'need to know' model for transactions is employed, meaning that entities who are not party to a transaction are not in general aware that it happened.
- One way in which uninvolved parties can become aware of third-party transactions on Corda is if they subsequently take ownership of an asset that has passed through the hands of somebody else. The 'burn-mint' design for cross-partition transactions, as well as the use of separate application networks per partition largely mitigates this problem[44].
- To minimise risk of information disclosure to the platform operator (or Partition owner depending on deployment model), 'non-validating' notaries are used, which means the notary operator learns the information needed to confirm a transaction and avoid double-spends, but no more.
- Finally, all Corda transactions are encrypted in transit.

Performance and Latency

There was no stated throughput in transactions per second (tps) requirement for the Experimentation Platform. However, we observe that many real-time payment systems today typically operate and 2000-5000 tps, and it is conceivable that the platform may need to ultimately support levels similar to the 100,000 tps contemplated by the Bank of England's Digital Pound Technology Working Paper[45].

No formal testing of the performance of the Experimentation Platform was performed. However, from a comparison of architectures, it is likely that the performance characteristics of the Experimentation platform would broadly match Open Banking, although they do not, as yet, match those of the Cards or Faster Payments systems.

44 However, in scenarios where an underlying record contains some unique information such as a serial number that needs to be preserved across transfers, the burn-mint approach may introduce complexity.

45 https://www.bankofengland.co.uk/-/media/boe/files/paper/2023/the-digital-pound-technology-working-paper.pdf

Similarly, there were no formal latency requirements for the Experimentation Platform. However, it was designed with other platforms in mind. While only limited latency testing was performed, it was measured in the payment card Business Application. The response time in that application was about 4.8 seconds, which is much higher than would be required for production usage, so optimisation or alternative design choices may need to be explored.

API and Orchestration Layer Performance

The API and Orchestration Layer was designed such that it was not likely to be a bottleneck, could be run with multiple instances and made idempotent. However, it was unavoidable that many capabilities exposed by the API necessarily involved several steps within the Orchestration Layer, each of which in turn may require invocations of underlying ledgers, the performance and latency of which the overall solution cannot control. To mitigate this risk, two steps were taken:

- The API, wherever possible, works asynchronously, so that calling applications are not blocked unduly and can retrieve the results of operations at their convenience;
- Certain data, such as the status of some locks, was cached by the Orchestration Layer.

Ledger Performance and Scalability

Corda's 'out of the box' transaction processing model was employed, wherein each transaction is only shared by and/or verified by parties with a 'need to know'[46]. This means that each (virtual) node only needs to be sized and configured for a subset of the overall transaction volume being processed across the entire deployment.

Furthermore, a direct consequence of the federated network of networks design is that each partition, including the notary, in the tokenisation platform can be scaled independently, with each partition owner incurring only the cost necessary to support the transaction throughput for their customers. However, the orchestration layer (whether embedded in the Tokenisation Platform or reused from the overall Orchestration Layer) would need to be sized for anticipated cross-partition operations.

The Experimentation Platform was designed to support a modest number of full participants on each partition, each represented by a virtual node, and the solution was performant when used. However, to support full retail volumes, especially in the case where individuals are not expected to interact directly with the ledger, it may be more appropriate to model individual users as accounts within a virtual node (or even held off-ledger) rather than giving each person a full ledger identity. And the extent to which banks' corporate customers, for example, desired to fully participate in a partition may be a key consideration when deciding on the most appropriate overall shared ledger architecture for future phases.

If a different ledger architecture is considered for the future, especially one that has a higher degree of data sharing or one where a common consensus process is utilised, it is important to ensure each node is sized for the full set of transactions it will be exposed to and that the shared notary (if Corda is used) be appropriately scaled.

46 Note that, in some Corda deployment models where assets move freely in a 'bearer' style, the 'need to know' can entail the need to verify some historic transactions, with an accompanying increase in the load on each virtual node. However, the model utilised on this project did not rely on these long 'chains'.

Availability and Resilience

As discussed in the Architecture Decisions appendix below (from page 82), one advantage of a more fully shared architecture is that participants could potentially recover from their peers in the event of a catastrophic outage. This possibility was not demonstrated in the Experimentation Platform, however. Should it become a firm requirement, a reliable solution would require certainty that *all* relevant data (on-ledger, some off-ledger and potentially also in the orchestration layer) was securely and definitively replicated to at least one other peer. This would likely pose an overhead, albeit modest.

However, support for high availability is a key capability of all layers of the architecture. For example, at the Tokenisation Platform, the recommended architecture consists of a series of stateless 'workers' performing work for any identities hosted on that infrastructure, managed as part of a Kubernetes cluster.

06:

Implications for Future Phases

Implications for Future Phases

The design of any future-state production solution would likely differ to the architecture of the Experimentation Phase. This is because:

1) The design of the overall experiment was necessarily simplified, in some significant ways.
2) There are at least two ways to meet the requirements of the Tokenisation Platform, with different trade-offs, but the Experimentation Phase only permitted the technical exploration of one of those models.
3) The project has yielded important insights in all dimensions (legal, business, technical) that would inform a future-state design.

In addition, it is possible that a phased delivery approach would be necessary.

In this chapter we summarise the key business questions and requirements that would have most significance for the design of these platforms or decision-making with respect to roadmaps. The chapter concludes by discussing several concrete pieces of future work that could help de-risk subsequent phases or inform the answers to these questions.

API and Orchestration Layer

Authority of the API and Orchestration Layer

In the Experimentation Phase the API and Orchestration Layer authenticated to the underlying ledgers and, from a technical perspective, how those ledgers chose to interpret messages (instructions versus requests) had no impact on the design. However, for a production implementation, a firm decision would need to be made: would the orchestration layer propagate evidence of intent, such as signed messages passed into the API, from its users or would it adopt a similar model to the Experimentation Phase? And, if the latter, would the messages be considered instructions or requests?

Deployment Model for the API and Orchestration Layer

The API and Orchestration Layer was deployed as a traditional centralised architecture. However, one can imagine a model were there is no central infrastructure and these layers are, instead, implemented in a more peer-to-peer model by the participants themselves. Such a design has not been explored, and the attractiveness or otherwise would likely depend on the legal treatment and structuring of any platform operator, but if such a model is contemplated it would be highly architecturally significant and so a decision of this sort should be made very early in any programme.

More generally, the amount of state managed by the Orchestration Layer means that observability of the system by participating banks is likely to be an important requirement that must be satisfied by a production architecture. Similarly, the resilience and recovery design should take into account the possibility that state is held at several layers of the architecture (eg in both the Orchestration Layer and Tokenisation Platform).

Identity Management

The Experimentation Phase made a conscious decision to avoid an 'enrolment' process whereby bank customers were 'onboarded' to the system and given an identifier. Avoiding this step removes a blocker to adoption, and the project has demonstrated that a 'no-enrolment' model is technically feasible. However, we demonstrated that this approach can complicate the API, and could impose burdens on those interacting with the system in terms of the information they need to supply and the extent to which details of underlying ledgers can 'leak' into the interface. Therefore, a key early decision to make is whether a future-state API would require enrolment or not and, in a related decision, whether the operator of a future platform would need to implement a PKI. Furthermore, if an enrolment step is considered for future phases, the design should allow for the possibility that one or other underlying ledgers adopts a system of one-time aliases.

Tokenisation Platform

Shared Ledger versus Federated Network of Networks

The first and most fundamental question is which design to select for the future-state Tokenisation Platform. As is discussed in some depth in the Architectural Decisions appendix to this report, a federated network-of-networks model was explored in the Experimentation Phase (see page 81).

In summary, both architectures can enable the creation of a multi-issuer, programmable tokenisation platform, with the requisite levels of privacy. If the business requirements favour greater control for issuers, a need to scale to retail volumes, and have relatively lower demand for rapid, iterative deployment of composable business logic then a network of networks model may be more appropriate. If, by contrast, the benefits of true atomic settlement across asset types and issuers are paramount, potentially with more rapid evolution of business logic and without as strict a need for privacy, as may be the case for wholesale markets, then a more shared architecture could be more appropriate.

Tokenisation of Central Bank Money

A key policy question to clarify before the design of a productionised Tokenisation Platform can commence is whether tokenised central bank money would be available on the platform. If so, it would raise the relative value of a shared ledger architecture in that the possibility of atomic settlement of payments entirely across a Tokenisation Platform. However, if the likelihood of tokenised central bank money is lower, the platform would necessarily need to integrate with a central bank money settlement rail (such as RTGS) and so the relative importance of cross-ledger synchronisation would rise.

Multi-asset versus money-only

A second decision relates to whether the Tokenisation Platform would host other assets in addition to money. If so, it is possible that the the atomicity enabled by a shared consensus layer more closely matches users' expectations.

Wholesale focus

The design of the Experimentation Phase contemplated the use of the Tokenisation Platform for the execution of retail payments, with the associated privacy requirements. Should the Tokenisation Platform be targeted at predominantly wholesale use-cases, however, then some of the privacy requirements could potentially be relaxed, which would serve to open up the design space with respect to shared architectures.

Ledger Access

The Experimentation Platform was designed to allow sophisticated bank customers, in principle, to operate in one or more partitions as full participants. This led, in part, to the federated 'network of networks' design. Should this not be a requirement for production and if it is preferable for bank customers to interact via interfaces managed by their bank, a more shared architecture[47] may be appropriate.

End-to-End

Locking

Many models for locking were explored in the Experimentation Phase, including some sophisticated 'multi-lock' models, with associated implementation complexity, including subtle interactions with the various tokenisation accounting models (pooled, shadow, etc). The design of a production solution could be somewhat simpler than the Experimentation Platform once a subset is selected for implementation.

Network Infrastructure

Aspects of the final design of the API would depend on whether it would be accessible over the public internet or only via a private network.

Programmability and Extensibility

Deployment Model and Associated Support Infrastructure

A fundamental part of the vision for a platform for innovation is that it be extensible. In other words, it must be possible for new functionality to be added. For example, the fourteen Foundational Capabilities explored in the Experimentation Phase are unlikely to be sufficient and so extensibility to enable the addition of more must be possible. This means the future-state design must not only be extensible at a technical level but the associated operating model must support it.

One starting point might be to consider how new capabilities are added to many permissionless blockchains. Anybody can add new 'smart contracts', which can be surprisingly complex and capable, but upgrades to the core protocol are far more closely controlled. Given that this platform would be a piece of critical infrastructure, potentially transacting vast amounts of money, it is highly unlikely that arbitrary parties would be permitted to deploy new logic without any controls. However, a model whereby the equivalent of a 'hard fork' were required to modify how a particular type of lock was implemented would be equally unsatisfactory.

It is likely that a workable model would be one whereby it is very easy for innovators to develop and test potential capability enhancements in a safe yet realistic environment, but where the actual deployment is subject to a well-documented, well-resourced review and testing process. This means that the associated organisation would need to consider competencies such as 'developer relations' and the maintenance of high quality documentation and development kits.

47 That is: a model where the Tokenisation Platform is implemented as a single shared network between banks, each with their own node.

It is important to note that the review and deployment model selected has a bearing on the design of the Tokenisation Platform. The more rapid and iterative deployment can be the more naturally it would fit with the way a fully shared 'composable' ledger design operates. If a more deliberative process is required or there is a requirement that business logic be deployed incrementally (e.g. tested first in one partition before rolling out to all) then the more compartmentalised federated network-of-networks model may be more attractive.

Threat Model

Irrespective of the deployment model chosen, it is likely that a platform that processes large amounts of money, and is capable of programmable extensibility would be a very attractive target for hostile actors, who would have a strong incentive to have malicious code deployed into the system. Experience suggests that 'defence in depth', for example by through implementing multiple layers of protection, is good practice. An example of one such layer that should be considered is a sandboxed execution environment hosting a carefully defined, expressive but deliberately limited 'domain-specific language'.

Recommendations for Further Work

Privacy model

As discussed earlier, the data privacy concept explored in the Experimentation Phase was based in part on separating data entering the API into an encrypted portion, readable only by one or other participant but not by the operator of the API and Orchestration Layer, and a portion that was visible to the API and Orchestration Layer. The idea was that, in many cases, the orchestration layer does not need to see personal data in order to satisfy requests, and so such information can be placed in the encrypted portion.

This worked well in most cases. But a number of situations were encountered where the API and Orchestration Layer, or an existing downstream ledger, needed access to information in the encrypted portion. This suggests that further design work is required to refine this concept. Note that it may prove to be difficult to eliminate all personal data from being visible to the platform in all cases.

Identity model

The implementation of the Experimentation Phase was considerably simplified by adopting an 'authority' model in which the orchestration layer made *requests* to underlying ledgers (eg to request approval from a customer to make a payment) rather than sent *instructions*.

However, such a model may not be workable at scale in production and it may be necessary to adopt a different model. For example, one design may entail instructions sent to the API being digitally signed in such a way that the underlying ledgers can verify that requests are already suitably authorised. Alternatively, a model where the platform operator is authorised to issue instructions directly to underlying ledgers could be considered.

In either case, further design work would be required, and experimentation with a prototype of the proposed solution would likely be beneficial in identifying any nuances associated with particular underlying ledgers.

Experimenting with a fully shared ledger

The Experimentation Phase found that the functional requirements of the project could be met with the federated network-of-networks model, and identified potential advantages of the approach. It also identified capabilities that might be easier to achieve with a fully shared ledger.

The postulated benefits included the ability to provide atomicity across partitions, the potential to increase resilience through recovery of data from peers, elimination of 'burning' and 'minting' between partitions, and the possibility of faster, 24/7 settlements. The report speculated that the high degree of infrastructure commonality required to achieve this may be difficult to achieve at scale.

A subsequent phase that implements and evaluates this alternative ledger model, perhaps incorporating a range of asset types and not just money, could be an extremely valuable complement to this Experimentation Phase.

Tokenisation Model

The Experimentation Phase analysed three distinct ways in which tokenised assets could be represented within existing bank systems and/or reconciled with them. The 'shadow' model potentially simplified operations and integration work for banks but created a need to maintain consistent records between the Tokenisation Platform and participating banks. The relative costs of each model could be analysed more deeply in a future piece of work.

Note that, for the Experimentation Platform, this synchronisation was implemented by the orchestration layer. Future work should explore the most appropriate way in which this synchronisation should be achieved in production.

If the Experimentation Phase model were to be pursued in production, further design work should be commissioned to identify opportunities to limit the amount of implementation detail (such as the details of the shadow accounts) that 'leaks' through the API.

Netting of low-value payments

As discussed above, the Experimentation Phase explored a model for executing low-value transactions without requiring simultaneous settlement in central bank money. This demonstrated that the architecture could support the separation of these actions. However, the project did not design or implement a netting arrangement such that a batch of low-value transactions could be settled by one netted Central Bank money movement. Should deferred-net settlement be a requirement for a Productionised system, design work would be required in this area.

Extensibility

The platform was designed to allow extensibility at multiple levels, as discussed above. However, during the excecution of the project, testing of this capability was restricted to the API Layer, where third-party Business Application developers demonstrated that the API could meet their needs. A future phase, perhaps in the form of a 'hackathon' could validate the extent to which the other layers of the platform meet the extensibility objective.

Integration with existing payment schemes

The Solution Design chapter, above, described the challenges of integrating the 'request to pay' concept with how certain existing payment systems operate. If a different 'authority' model is adopted, as discussed in the section above, then this may not be an issue. However, should the Experimentation Phase model be adopted for a production implementation then specific design work for integration with systems such as FPS or Fnality should be scheduled, including consideration of the role of extensibility in facilitating connectivity scenarios.

Data and process model

The opportunity to reuse or adapt existing standards at different layers of the architecture could be explored in future work.

For example, the Experimentation Phase API utilised JSON and was loosely modelled on that developed for Project Rosalind. Further work should consider whether a range of options should be offered, to minimise integration cost for existing applications, as well as exploring whether ISO20022 (or an extension to support requirements such as locking that are not currently part of the specification) should be adopted. As part of this analysis, alternatives or additions to the webhook-inspired asynchronous communication model, such as a messaging model, should be evaluated

At the orchestration layer, existing and emerging process standards such as CDM[48] could be explored.

Technical Analysis of Existing Initiatives

The Experimentation Phase explored a number of distinct concepts which can inform business decisions around implementation roadmap options. However, there are existing services or initiatives which provide relevant context and which should be studied as part of future work. For example, the emergence of 'API Aggregators' for Open Banking could provide insights into the needs of API users. For example, what functionalities could be provided that none of these existing API aggregators already provide, and which of those functionalities could simply be added to an existing API aggregator vs which of those functionalities would require a tokenised deposits platform?

48 https://www.icmagroup.org/market-practice-and-regulatory-policy/repo-and-collateral-markets/fintech/common-domain-model-cdm/

07

Appendix: Architectural decisions

Appendix: Architectural decisions

This appendix considers the key questions of architectural significance for the project. For the API and Orchestration Layer, a wide range of relatively narrow questions arose, such as how consumers of the API should interact with it. For the Tokenisation Platform, the questions generally concerned implementation: to what extent and in which ways should the ledger be shared among participants? We also consider some of the questions that influenced the overall end-to-end architecture.

API and Orchestration Layer

Introduction

The key architectural questions influencing the design of the API and Orchestration layer were, in many cases, questions about requirements. For example, how should messages from the orchestration layer to an underlying ledger be interpreted? As an instruction? Or as a request? Similarly, should an individual be 'enrolled' into the system and allocated an identifier before being able to use it? Or should they be able to use it without any prior registration? The answers to several of those questions were of architectural significance and are explored below.

API Layer: Avoiding a need for Pre-Enrolment

When implementing an API that provides one interface in front of multiple implementations it is often desirable to prevent the details of those implementations 'leaking' through the API.

For example, a potentially elegant design for the API could simply present generic "Account ID"s as one of the fundamental entities managed by the system, and which could map to accounts or wallets on any of the underlying ledgers. Under such a model, the bulk of APIs could be defined without reference to the form of money or type of ledger being operated on.

A first version of the design contemplated this model. However, to support it, an 'enrolment' step would be required for each user and service user. For example, the API and Orchestration Layer would need to create a mapping from the putative Account ID to the actual ledger and account or wallet to which it related.

Enrolment of individual users, however, would place a large burden on the institutions, potentially harm the business case, possibly inhibit adoption, and might result in the orchestration layer holding more personal data than strictly needed for the processing that it handles.

As a result, enrolment was removed from the design for the Experimentation Platform. A consequence of this was that those calling the API needed to provide the information that the orchestration layer needs to service the request, such as underlying ledger and associated account numbers. The specific format used to signal this information is described in the Solution Design chapter above.

API Layer: Communication Style

One of the design priorities was to deliver an API that was easy for developers to consume. One approach taken to achieve this was by avoiding, wherever possible, a mixture of synchronous and asynchronous calling styles. Instead, given that many of the calls could potentially take some time to complete (e.g. complex workflows requiring approval), the API was designed to work asynchronously, except in some specific cases such as balance checks where a synchronous model was clearly superior.

This raised the question of how asynchronous responses to requests, as well as unsolicited messages such as advices about inbound payments or incoming payment requests, should be made available to API users. Options including sockets, file transfer, message queues, polling and webhooks were considered.

Webhooks were the most natural approach given the REST decision for the API itself. However, experience from Project Rosalind demonstrated that it can be surprisingly onerous and time-consuming for allow inbound messaging to some financial institutions' infrastructures.

As a result, a "virtual webhook" was implemented. That is: an API endpoint that, when polled, would send the data that would have been sent via the webhook. This 'mailbox'-style approach had the benefit of conceptually simplicity, but some developers found it cumbersome to work with.

Orchestration Layer: Meaning of 'Orchestration'

At an abstract level, the API and Orchestration Layers can be thought of as having a purely technical function: taking a request message in one format from one source and sending messages of different formats to other systems.

However, the meaning of the messages has significance for the design. In particular, when the orchestration layer sends a message to an underlying ledger, should that system interpret the message as an *instruction* to do something, or as a request to be considered for approval? If the former, should the underlying ledgers proceed on the 'say so' of the orchestration layer? Or does some evidence need to be provided to convince the underlying ledger that it has authority to proceed? If the latter, to what extent should the orchestration layer be responsible for obtaining such authorisation, and what might be the architectural implications of any latency or routing that arises as a result?

Three alternative design approaches were considered:

- The orchestration layer could pass *signed* instructions from API users to the back-end ledgers and other parties, thus providing evidence of intent from the originator
- At the other extreme, the orchestration layer could act on behalf of the member institutions or even end-users, directly creating authorised instructions for the back-end ledgers, after suitably validating and authenticating requests arriving through the API

- Following existing industry 'request to pay' models, the orchestration layer could call in to the underlying platform, with the message interpreted as a request to be forwarded to the relevant institution or customer to action or approve. Depending on the nature of the ledger, this may result in that institution injecting an instruction or signing a proposed transaction, say, but the outcome is that the orchestration layer has requested the relevant institution to perform the appropriate action, meaning the institution is the decision maker.

The first option is problematic because the most elegant approach would require modifications to all the back-end systems to enable them to recognise these new data structures and passed-through credentials. But if those calling the API instead used existing credentials for the back-end ledgers when calling the top-level API we risk ending up with a convoluted solution and difficult developer experience.

The second option was not selected for the experimentation platform as it either required delegating an unrealistic amount of authority to the platform operator or would require wholly new authorisation flows and legal mechanisms to be designed, possibly requiring all members to implement another new form of real-time authorisation that would likely result in high implementation costs for members.

We therefore selected the third option, to protect existing investments in interfaces into back-end ledgers and to avoid having to assume that other systems and schemes would be modified.

To accommodate the 'approval' concept, support for encrypted and digitally signed bilateral messages (which could be based on ISO20022 in the end state) between institutions was added to the design so that contextual information underpinning a request could be shared with the associated institution in order to make an approval decision, without revealing that extra information to the platform operator.

Orchestration Layer: Privacy

The privacy model used in the orchestration layer was designed to avoid the orchestration layer having to hold or process personal information or sensitive information such as payment purpose.

The key idea was a scheme whereby messages passed into the API would consist of two parts. The first part is for the API and Orchestration Layer, and consists of very limited data: originator and beneficiary account numbers, money type, amounts, lock types, etc. This is encrypted in transit but is a cleartext payload on receipt by the orchestration layer.

The second part of the message is envisioned to be for delivery to the counterparty, or other involved parties, and is end-to-end encrypted so that only the intended recipient can read the data. This was to be achieved by having the sender encrypt the message based on the public key of the recipient.

The model is intended to minimise, as much as possible, the amount of personal data to which the platform operator has access, while providing the ability for participants to exchange information where required. It implies the need for the operator to implement (or consume the services of) a Public Key Infrastructure (PKI). For the Experimentation Platform a PKI was not used.

A finding of the project is that this model, if fully implemented as described, would prevent some orchestration operations from having access to the information necessary to succeed. This means that further design work may be required in this area. See page 73.

Orchestration Layer: State Management

The nature of an orchestration layer, which necessarily needs to coordinate a potentially long-running sequence of steps, is that state must be managed. Furthermore, the objective of delivering functional consistency created information that sometimes needed to be stored by the API and Orchestration Layer because an underlying platform lacked the necessary features. For example, the use of escrow for platforms that cannot support locking requires information about the state of the escrow arrangement to be maintained by the orchestration layer.

The approach taken by the project was to minimise the amount of state held, with the following principles used to guide the design:

- The orchestration layer will keep state for functionality that is needed for functional consistency, if such data is not available in back-end systems.
- The orchestration layer will keep end-to-end transaction state persisted to the extent necessary to support queries, and will hold notification messages in case members need them to be replayed. In the Experimentation Platform we hold undelivered notification messages that are waiting to be polled for, but in a production system these would be delivered in real time.
- Otherwise, the orchestration layer will not mirror state that is held in other systems. This includes balances, lock statuses and internal transaction states in back-end ledgers.

The Experimentation Platform conformed to these principles except in one place: Rosalind lock statuses are cached for simplicity of implementing workflows. This was a tactical decision to meet project timescales, and this does not imply that a production system should do the same.

Tokenisation Platform Architecture

Introduction

In this section we discuss the rationale for the design chosen for the Tokenisation Platform and explore alternatives.

The design implemented for the Experimentation Phase was a federated 'network of networks' model. A single multi-institution Corda deployment, spanning several clusters modelling self-hosted banks and one centrally hosted installation, was deployed. For each issuer, an 'Application Network' was configured, representing that issuer's 'Partition', with a node for the issuer and nodes representing some customers. This is depicted in Figure 25, on page 88.

An alternative design could have been to instead deploy a single network, with a node for each participant, and any common/central infrastructure (such as notary, identity infrastructure) managed by an appointed entity. This model is depicted in Figure 23, on page 86. The alternative design would meet several intuitive notions of what it means to be a 'shared ledger':

- any data that two or more parties need to have in common would be reliably replicated between them, so we achieve *shared data*;
- the participants would be users of common infrastructure (eg consensus nodes, identity management, and so on), so we achieve *shared infrastructure*; and
- any code (such as smart contracts) deployed to the network would be available to all, in principle, so we achieve *shared business logic*.

However, a single network design has some potential drawbacks. Some reasons why the fully shared architecture *may* not always be the optimal one include:

- A high degree of infrastructure commonality means participants must submit to a greater degree of pooled sovereignty than may be acceptable to them. This includes the potential need to cede control of transaction confirmation to the common operator.
- The architecture would need to demonstrate that is can scale to retail volumes.
- In non-greenfield implementations, where some participants have already made technology choices or deployed their own networks (such as a tokenised deposit network for the use of their own customers), there may be a preference to 'federate' existing networks into a network-of-networks rather than migrate to a new architecture[49].

None of these objections are decisive, but they demonstrate that a range of implementation options is possible. An architectural decision does therefore need to be made. In the following sections, we examine three questions that informed the experimental design choice:

- What degree of data sharing is desirable or permissible, and between whom?
- To what extent should the owner of each partition be able to operate independently from its peers from a technology roadmap perspective?
- How should cross-partition atomic updates be achieved?

The question of 'programmability' is not included in the list above. This is because all architectures we will discuss are capable of delivering the same outcomes at a functional level. However, some models have (such as if code needs to be kept private to a specific partition or deployed once for all parties). We make some observations about this below.

Degree of Data Sharing

The Tokenisation Platform manages records of bank liabilities (equivalently, customer assets), and the thesis is that these representations are easier to manipulate than existing methods. These core records, which may be records of account balances or of the existence of some tokens, are what drives decision-making when thinking about data sharing. And the fundamental question is: *for any given record, who should be in possession of it?*

Consider the following scenario:

- Customer Alice of Bank A holds a balance of tokenised deposits
- She wishes to make a payment to Customer Bob of Bank B
- Bank A wishes to settle the transaction by making a transfer of tokenised wCBDC to Bank B

So we are concerned with three partitions: the partitions of Bank A, Bank B and the Central Bank.

And our objective is to:

- 'Burn' some of Alice's tokens in Bank A's partition,
- 'Mint' some tokens for Bob in Bank B's partition, and
- 'Transfer' some tokens from Bank A to Bank B in the central bank partition.

49 However, it would of course be possible to continue running the existing system and enable the API and Orchestration Layer to integrate with it instead.

There are potentially four records of interest: the record of Alice's balance, the record of Bob's balance, and the records of Bank A and Bank B's balances with the central bank.

As a starting point, we observe that the following should hold for any architecture:

- Alice and Bank A should be able to see and to agree on Alice's balance at Bank A.[50]
- Bob and Bank B should be able to see and to agree on Bob's balance at Bank B.
- Bank A and the Central Bank should be able to see and to agree on Bank A's balance at the Central Bank.
- Bank B and the Central Bank should be able to see and to agree on Bank B's balance at the Central Bank.

A maximally private design would ensure the above and no more[51].

However, some ledger architectures result in a greater degree of data sharing, with some corresponding benefits, and there are three scenarios for greater data sharing that we should consider:

- Given this transaction involves Alice, Bob, Bank A, Bank B and the Central Bank, is it permissible that all five of them come to learn each other's balances? We assume for retail scenarios that this is not appropriate.
- Alternatively, is it permissible for all parties within a given partition to see everything that is happening in that partition? By way of example in this case, it would mean that Bank A and Bank B would see each other's activity and balances in the central bank partition. Again, this is unlikely to be appropriate for retail scenarios but may be acceptable for some wholesale scenarios.
- Alternatively, what if we allowed all parties to see *everything*, even records in which they were not a participant at all? This is similar to how an 'out of the box' Ethereum-style architecture operates. It clearly works, as the permissionless Ethereum network shows us. However, we assume again that the privacy implications in a regulated UK retail payments context would not be acceptable[52].

The model adopted for the Experimentation Phase was one in which only the parties to a ledger record (typically the issuer and the owner) had access to it – a 'need to know' model. As the above discussion has highlighted, other models are reasonable, and one important reason to consider broadening the distribution of records on the ledger might be if it could make the overall system more resilient.

50 An interesting observation from the project is that not all participants had assumed the possibility that bank customers would be represented directly on the ledger by having their own node. Should this not in fact be a requirement then this might argue in favour of the more simple 'one single network' model.

51 For brevity, we do not consider instructions and other transaction information in this analysis. In cases where the distribution of an instruction may be wider than the distribution of the balance records it references, the set of parties listed above may need to expand. However, we observe that the present-day technique of splitting such messages (eg introducing a cover payment) already provides a model for how to limit unnecessary data sharing in such situations.

52 Advanced cryptographic techniques such as zero knowledge proofs may have the potential to address these issues, but their evaluation was not in scope for this project.

State Recovery from Peer Nodes

One important postulated benefit of a shared ledger architecture is the opportunity it affords for one participant to recover data or other state from *another* participant, in situations where one of them has suffered a catastrophic outage. However, the viability of this concept depends on the data sharing model used.

For the Experimentation Platform, where record sharing was limited to the parties involved, there are situations where there are no other banks in possession of some records and, for those records that are shared with other institutions, it is likely that full recovery would require the participation of many or all other network participants. By contrast, a model based on full data sharing amongst all parties could potentially allow one firm to recover with the participation of only one other peer.

This leads us to our first key finding: should banks wish to use a shared ledger as the basis of a backup or recovery strategy this must be specified as a key requirement up-front, and the design should allow for it. In essence, there may be a conflict between privacy and easy data recovery, and the viability of the recovery concept is therefore primarily a business and policy question.

To meet a peer-recovery requirement, the following alternative options might be feasible:

- A full-data-sharing architecture could be an enabler of 'peer recovery' provided the non-technical implications of full data sharing could be addressed from a privacy perspective.
- One or more parties could be added to the 'need to know' list for data records, ensuring that there is always a minimum number of nodes guaranteed to have a copy of any given record. In this case, SLAs may need to be agreed with those parties, as well as contractual obligations related to data protection.
- A well-regulated 'stand-in' bank or other suitable entity could, with appropriate safeguards, receive replicated data for all banks and offer recovery services.

In a future phase, techniques to maintain data privacy whilst providing easy data recovery can be explored. One such technique could be where each node encrypts its data using its private keys before sharing with some or all other nodes, or shared with a platform operator 'master node' which can on an as-needed basis give the slice of the data owned by a participant institution.

Finally, it is important to reflect on the limitations of this concept, even in the full-data-sharing case. The data that can potentially be recovered from one's peers is the data managed by the ledger, and no more. There is invariably a large amount of off-ledger data that must be managed and protected.

Consider a tokenised deposit design where a unique token record is created for each payment a customer receives, each with its own address or key. This is a common privacy-enhancing technique in permissionless blockchains. By design, there is no record on the ledger that tracks which accounts belong to which customer; it would defeat the privacy objective otherwise. So, if one were to suffer a catastrophic data loss, it would not be enough to recover the ledger data; one would need to recover the off-ledger data that holds the mappings between ledger accounts and customer identifiers.

Infrastructure Commonality

The second key question is: to what extent should the technology used by each participant be standardised and/or operated as a shared infrastructure? Intuitively, we can assume the answer is likely to be: maximise sharing and commonality to the greatest extent possible, to benefit from economies of scale, reduce test effort, reduce code duplication, and enable the full feature set of the underlying platform to be exploited as much as possible, including any native support for atomicity.

However, maximisation of commonality is the same as minimisation of divergence, meaning each participant has less flexibility in their ability to conform to internal technology standards, upgrade or apply maintenance to their own schedules, and so on. Having a common technical implementation can create concentration and security risks[53]. Moreover, a single shared network may be unable to support all UK retail volume, although if it is used primarily for traffic that requires the unique features of a shared ledger then this need not be an issue in practice.

There are three broad approaches one can take:

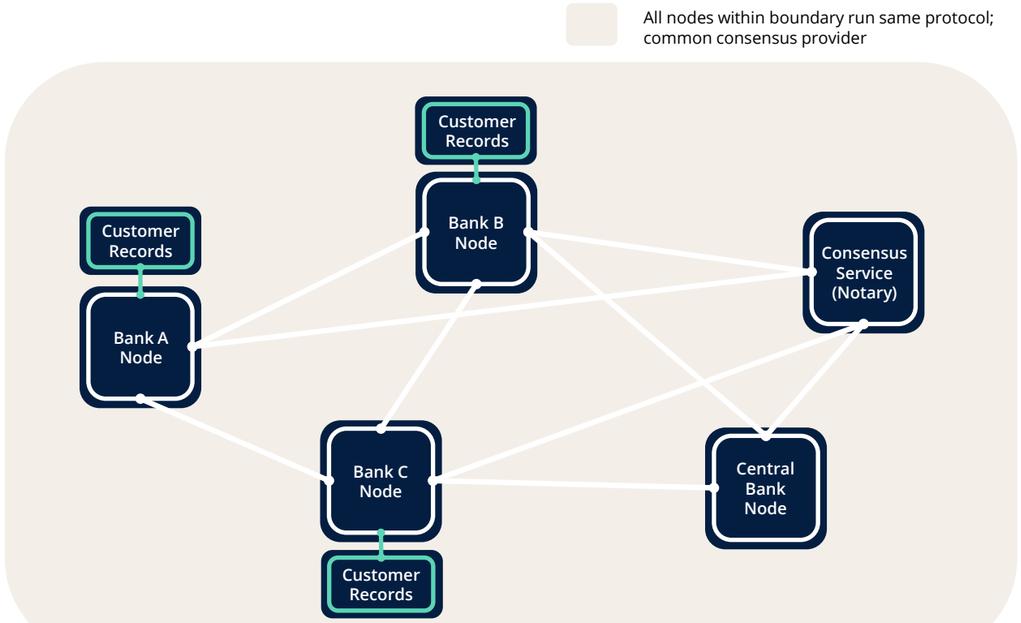
1. Fully common, shared infrastructure.
2. Fully decoupled partitions, accessed via API.
3. Multiple instances of a common platform.

1. Fully common, shared infrastructure

At one extreme we can imagine a single shared infrastructure, in which each node owner operates or utilises the same software, within clearly defined version bounds, configured in a common manner, enlisted into a common governance, security, identity and operational model, forming a single shared DLT network. The 'mainnets' of the major permissionless networks work in this manner and it is depicted in Figure 23, below.

53 Investment in defining a protocol that could support independent implementations to develop over time could mitigate these risks.

Figure 23. A fully common, shared infrastructure, in which each bank has a node, there is a single consensus process, and the Central Bank is also represented



This model has several advantages:

- Conceptual simplicity: all major decisions are made once, by the central operator, freeing each partition owner from having to each do this work.
- Organisational 'hammer': the fact that the approach permits almost zero opportunity for technical divergence can sometimes allow project teams to secure waivers from onerous internal standards if the business imperative to join the network is paramount, whereas designs that permit flexibility create the opportunity for internal IT teams to impose their rules[54].
- Potentially lowest overall technology cost: less testing, fewer version-version incompatibilities, and the ability to write business logic only once.
- Greater capability: the potential to fully exploit all the capabilities of the underlying platform, because everybody is running the exact same stack.
- Single network: asset owners only need to connect to a single network irrespective of which partition(s) they wish to interact with.

The key disadvantages of the model are the high degree of coupling it creates and the associated concentration and security risks. For example, nobody can upgrade to the next major version until *everybody* agrees to upgrade and everybody must implement the same network parameters and rules.

54 Of course, if the result is additional internal costs and increased perceived risk as a result of not aligning with internal standards then the business case for adoption could be weakened.

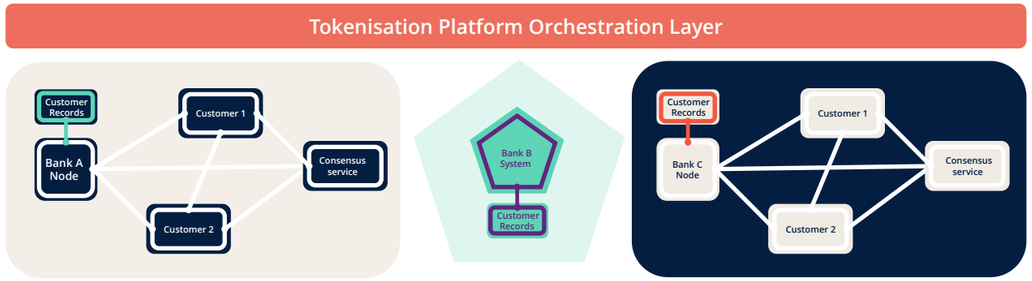
This need for 'extreme commonality' is a well-understood phenomenon of blockchains – the need for hard-forks in Ethereum is a consequence of this. It is a direct result of the need to ensure, with certainty, that nodes remain in absolute consensus with each other.

The anticipated scale of UK RLN, with potentially dozens of issuers and multiple asset types, means the coordination challenge could be significant, but it is a potentially viable approach.

2. Fully decoupled partitions, accessed via API

At the other extreme we can imagine each partition owner making entirely independent decisions for how they tokenise their liabilities and/or other assets, and having full control over how transactions are validated and confirmed. Nothing would be shared in this approach. For example, one bank might utilise Corda, while another might utilise Hyperledger Besu. And those with no intrinsic need for such a platform – e.g. to support their own tokenisation offerings for their clients – could even operate a traditional software architecture. Provided these institutions faithfully implement a common API, defined by the platform operator for the purposes of integration and cross-partition orchestration, they can participate in the network[55]. Some banks may choose to let their customers participate directly. This is depicted graphically in Figure 24, below.

Figure 24. The "fully decoupled partitions, accessed via API" model is one in which each partition could potentially be implemented with entirely different technologies, with no commonality and no common programming model.



This model has the advantage of maximum flexibility for each partition owner, but is likely to be more expensive overall, with higher testing overheads, and fewer opportunities for reusing code or programmability logic. It eliminates any possibility of utilising functionality specific to any one platform.

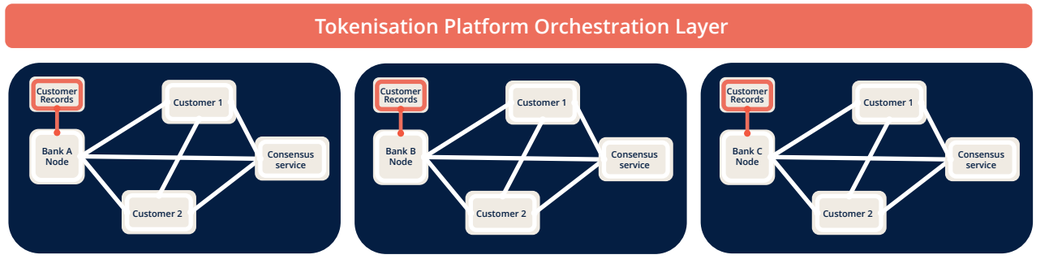
Perhaps more significantly, it would mean that any asset owners who wished to interact directly with the tokenisation platforms would need to install and maintain different clients for each bank they had a relationship with and handle the resulting complexity for themselves. This would be a significant and potentially show-stopping barrier if there was indeed a requirement to support direct access for customers.

55 See next section for a discussion of how transactions across partitions (in any of the models) can be achieved

3. Multiple instances of a common platform

A third option is a hybrid of the above two approaches. In this model, each partition is independent, as in the fully decoupled model above. But, unlike the fully decoupled model, the same platform is used, running the same business logic and smart contracts as provided by the platform operator. This is depicted in Figure 25, below.

Figure 25. Multiple instances of a common platform, unified under a common orchestration layer



In addition, it is assumed that the cross-partition orchestration is delivered by the underlying ledger platform rather than 'piggy-backing' on the broader API and Orchestration layer, such that the Tokenisation Platform is entirely self-reliant for all operations that happen across it.

The practical consequence of each partition being technically independent is that there can be more tolerance for divergence *between* partitions in terms of upgrade schedules, specific versions and partition-specific network parameters. But the use of the same platform retains the design benefits of a common programming model and ability to share code.

From the perspective of *owners/holders* of assets in the partitions who wish to transact as full participants, this model would allow owners to have a presence in all relevant partitions (their own 'virtual node' in each partition, in Corda terminology), while managing their virtual nodes from a single infrastructure.

The use of the same platform for each partition creates a similar 'concentration' or security risk to the fully shared model, but this is mitigated to some extent by each partition owner having the freedom to apply maintenance or upgrades on their own schedule. As a 'network of networks' model, it does not provide a cross-partition atomic settlement option, but see 87 for a variation on the architecture that might do so.

This is a plausible model for some scenarios but has not been extensively explored in prior experiments. The project took the opportunity to explore new ground and adopted this design for the Experimentation Phase[56].

Settlement of Tokenised Deposit Transactions

A hypothesis of the project was that a multi-issuer tokenised deposit platform could facilitate faster, potentially instant and 24/7 settlement of payments between customers of commercial banks.

56 However, for expediency in the Experimentation Phase, the broader API and Orchestration layer was used for cross-partition operations. This is discussed in a later decision.

There are three ways this could be achieved:

- If wCBDC is available natively on the Tokenisation Platform in a central bank partition then it would be possible for the Tokenisation Platform to coordinate the settlement across the three partitions (sending and receiving banks, and the central bank) without any reliance on any other infrastructure.
- If tokenised wCBDC is not available natively then the API and Orchestration Layer could coordinate the necessary activities between the Tokenisation Platform (commercial bank partitions) and non-tokenised wCBDC, RTGS or an omnibus account model.
- Alternatively, one could expand the role of the Tokenisation Platform such that it was able to connect directly to a suitable settlement platform and coordinate the necessary updates, without reliance on a broader API and Orchestration Layer.

The first two options were explored in the Experimentation Phase. The design considerations for the first option are discussed in the next section. We do not discuss the second option here as the questions arising are the same as for other settlements coordinated by the API and Orchestration Layer.

Atomicity within the Tokenisation Platform

At an end-to-end level for the Experimentation Platform, the API and Orchestration Layer is responsible for coordinating updates across systems, which is discussed later.

However, there are situations where *all* updates for a particular business transaction take place *only* across partitions of the Tokenisation Platform. In such cases, it can be possible for the Tokenisation Platform to coordinate these updates without reliance on the overarching orchestration layer, including in ways that may provide atomicity at a technical level.

The key question is: do all the partitions in the Tokenisation Platform use the same reference point (a 'notary' in Corda terminology) to determine whether confirmation has occurred in that partition? And if different partitions reference different notaries, how is settlement across partitions achieved?

The three distinct models are:

- One notary for all partitions.
- One notary per partition, with an additional common notary for DvP, to which assets can be temporarily assigned.
- One notary per partition, with an orchestration process used to enable a two-phase commit process across the partitions

The first two models provide a moment of atomicity that can potentially be referenced by the associated rulebook(s). The third model implements a process intended to ensure that if and when one technical component (notary) signals confirmation then it is assured that the others will do so too, but these events may not occur at the same instant[57]; they are synchronised, but not atomic at a technical level.

We now consider the advantages and disadvantages of each model.

57 This is, of course, also the model that the API and Orchestration Layer necessarily adopts given the heterogeneity of systems with which it interfaces.

One notary for all partitions

In this model we assume a single notary for the entire solution, such that the moment of 'confirmation' for each partition would be defined by reference to the actions of this notary. It is the model implied by the network architecture in Figure 23. This is the simplest model from a technical perspective, as it removes the need to worry about coordination of cross-notary updates. However, it has the potential to be a limiting factor on the maximum throughput of the network.

This model means that control over confirming transactions across the books of any given bank no longer rests exclusively with the bank concerned. This may not be an issue in practice, especially if the platform operator is responsible for the shared notary. But the potential need to support multiple notaries for performance reasons and/or to provide an option for partition owners to retain complete control means we should also consider options where multiple notaries are in use.

The obvious alternative to a design where all partitions share a notary is a design where each partition has its own notary (or other technical component used as the reference point for transaction confirmation). This design has the attraction that each notary could be operated by or for the bank whose partition it manages, thus ensuring that the component that determines finality is under the same control as the assets it is controlling. It has the advantage that each partition owner can configure their notary to meet their needs, from a scaling and performance perspective. It is the model we face with the network designs depicted in Figure 24 and Figure 25 above.

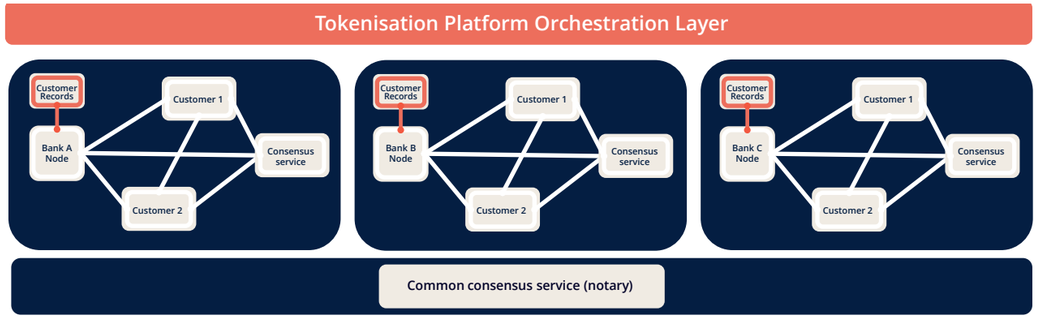
The primary disadvantage of this model is that for assets in different partitions there is no individual technical component that can be the singular reference point to establish a common point of confirmation to achieve atomicity. This may not necessarily be a problem from a legal perspective, but at the operational and technical level there does need to be a rigorous mechanism to ensure that if updates to two or more partitions need to happen on an 'all or nothing' basis then this outcome is indeed achieved.

There are two possible solutions. The first is to temporarily 'move' one or all assets to a common notary, which can then provide a simultaneous – atomic – confirmation for the multiple ledger updates, before moving the assets back to their original notaries. The second is to implement a locking and coordination protocol that ensures that, should one update occur, it is assured that the others will occur.

One notary per partition, with an additional common notary

The key idea behind this model is that each partition has its own notary, with the same rationale as above. However, there is an *additional* notary, most likely provided by the platform operator, to which assets can be temporarily reassigned. In this way, assets in different partitions that need to be transacted atomically can be moved to a common notary. Once on the same notary, the multiple transactions can be performed – and finalised – simultaneously. And the assets, with their new owners, can then be moved back to their 'home' notary. This is depicted in Figure 26, below.

Figure 26. In the 'additional common notary' model, an extra notary is available, to which participants in multiple partitions can temporarily 'reassign' records, in order that atomic transactions across assets associated with different partitions can be performed



We assume this process is coordinated either multilaterally by the participants or by utilising the associated platform's in-built workflow capabilities.

This model offers the scalability and performance advantages of the partitioned approach, with the possibility of atomic settlement of transactions that span partitions.

Potential downsides are the complexity entailed with 'moving' the assets, and any business and legal implications of the shared notary. As with the single notary model, the design requires all issuers of all assets to be comfortable with portions of their books being controlled, if only temporarily and in limited ways, by a third-party notary. And it requires the operator of that notary to be comfortable with taking on the associated risk. The latter is not a problem in principle but the need to support a wider range of assets and issuers, with the associated regulatory and legal analysis cost, while only being used for a small subset of transactions, may make the economics difficult.

However, none of these are decisive arguments against the approach. We did not pursue it for the Experimentation Phase but it remains a valid option for any future production state. Instead, we pursued a model that utilised the Tokenisation Platform's native in-built locking capabilities and which did not require assets to ever leave their 'home' notaries. We explain this further below.

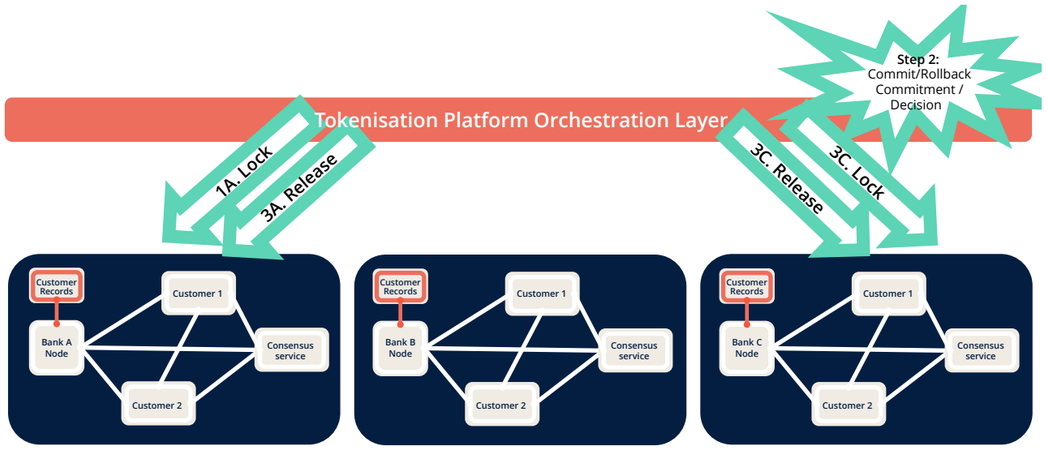
One notary per partition, with two-phase commit process across the partitions

The architecture chosen for the Experimentation Phase consists of a series of co-deployed Corda 'Application Networks', one per partition, each with its own notary. Each partition hosts virtual nodes for that bank's customers, and the bank is free to operate the partition in a way that best suits its business needs, and to scale it to match the size of its business. The technical component used as the reference point for transaction confirmation (the notary) is always run by or on behalf of the partition owner.

To synchronise operations across partitions we utilise an orchestration model, implemented by the platform operator, relying on the ability to lock tokens[58]. When one or more parties wish to perform an operation that affects two or more partitions, the model works as depicted in Figure 27, below:

- The request, suitably signed and authorised, is forwarded to the Tokenisation Platform's orchestration engine (Step 0, not shown)
- The orchestration engine calls each of the underlying partitions and informs them of the operation that is to be performed (Steps 1A and 1C). If the partitions validate the requests successfully and are capable of executing them, the associated records are locked and an affirmative response is provided.
- Should all partitions respond affirmatively, the orchestration layer records a decision to *commit* to the ultimate outcome (Step 2). If either partition had failed to respond affirmatively, the orchestration layer instead records a decision to rescind the locks.
- The orchestration layer then proceeds to tell each partition to *execute* (or rollback) the transaction that had previously been validated (Steps 3A and 3C)

Figure 27. Coordination of a transaction that impacts partitions A and C



This model eliminates the need for the platform operator or other party to operate a shared notary and define any required agreements. However, the way in which settlement is achieved is different: it is synchronised so that settlement across all ledgers happens *assuredly*, but not *simultaneously*.

In the first two models, two or more records are placed under the control of a single notary, and so any updates to multiple records all happen at precisely the same time: at the point the single notary signature is applied to the transaction.

58 Models that do not strictly require locking may also be possible but were not analysed as part of the Experimentation Phase.

However, in this model, there are two or more notaries and so the exact moments at which they apply their confirmatory signatures are likely to differ, if only by milliseconds. This is an unavoidable physical reality and the rulebook needs to specify how any edge cases should be handled. However, the corresponding benefit is that each partition owner retains control of their own confirmation process.

Decentralising the protocol

We can go further. Consider a different scenario: the exchange of two assets by two parties. We can use a similar model to the above in which the assets are locked in such a way that they can be released upon provision of a particular secret. The protocol is then structured such that confirmation of one transaction results in the release of that secret, which can then be used by the relevant party to ensure the other transaction occurs. In this model, the parties do not even need to rely on the availability or cooperation of the orchestration layer. This is roughly analogous to how 'hash timelock' protocols work in permissionless blockchains.

Discussion

A model in which all assets are controlled by a single notary is clearly the simplest, but raises questions of scalability and control. A model in which assets are controlled on a per-partition basis may be more realistic for the scale anticipated for UK RLN but it means cross-partition operations require special treatment. Introducing a common notary for this purpose is a viable option but raises addition questions related to moving assets from being under the control of the partition owner to the temporary control of some other party. For the Experimentation Phase, we explored a model where assets never leave their 'home' partition and, instead, we use the platform's native orchestration and locking capabilities to achieve synchronised outcomes. However, all three of the above models are viable depending on throughput requirements and appropriate structuring of the operator.

Optimisation for Experimentation Phase: Dependency on Orchestration Layer

The above section describes a model in which the Tokenisation Platform makes full use of an underlying DLT platform's capabilities, including an in-built workflow engine, where available, to implement cross-partition operations.

As a tactical optimisation to meet the timelines of the Experimentation Phase, the over-arching API and Orchestration Layer was used to coordinate cross-partition operations. In hindsight, it would have been better to explore the use of an orchestration layer associated purely with the Tokenisation Platform.

Programmability and Extensibility

A shared ledger can provide a single programming model for the definition of tokens and their behaviours, and for business logic that coordinates operations across assets or partitions. By contrast, a federated network of networks model introduces a separation between asset- or partition-specific logic, which can be deployed to the relevant partition(s), and coordination logic, which is deployed to an orchestration component.

A potential advantage of the shared ledger model is that the unified programming model may make it easier to deploy code units that can serve as building blocks for other code units, in a way that allows new functionality to be 'composed' by combining and augmenting existing functionality. This 'single computational and settlement venue' is similar to what is commonly done on public networks such as Ethereum.

In cases where new capabilities need to go through extensive design, review and testing, it is not clear that one model would be materially more productive or expressive than the other. However, if there is a requirement for rapid iteration and a need for simple 'composability' of any type of logic then the shared ledger approach may be preferred.

Ledger Architecture: Discussion and Conclusion

For the Experimentation Phase each partition was implemented as its own Corda Application Network, all deployed as part of a larger set of Corda clusters. In other words, this was a federated network-of-networks model. One of these partitions – wCBDC – contained a node for the Central Bank as well as nodes for multiple Commercial Banks and so was itself a shared ledger within the broader federated network. However, it was only used for the settlement of wCBDC transactions during the Experimentation Phase.

Bringing it all together, we can categories the various requirements that may lead to one design option or the other as follows:

Requirements satisfied by both architectures:

- Creation of a multi-asset, multi-issuer Tokenisation Platform
- Programmability
- Transparent and cryptographically secured record of transactions
- Settlement across multiple asset types and issuers

Requirements that would point towards a 'federated network of networks' model:

- The tokenised deposit layer is expected to process retail-scale volumes
- It is more important for partition owners to be in control of assets in their partition than to achieve cross-partition simultaneous atomic settlement
- There is a stronger requirement for finer-grained privacy control
- Banks demand greater freedom over their technology stack than would be allowed on a single network
- Banks have deployed, or are likely to deploy, their own tokenised deposit networks, that need to be federated into a single architecture

Requirements that would point towards a 'single shared network' model:

- Volumes can be handled by a single network and associated consensus infrastructure
- It is more important to achieve simultaneous atomic settlement than to maximise the control wielded by each partition owner, and tokenised wCBDC will be available to facilitate this
- There is less demand for strictly narrow privacy or even an explicit requirement for increased data sharing, perhaps for resilience purposes.
- Banks are willing to deploy identical technology stacks (or protocols)
- There is a requirement that composability scenarios across multiple asset types be simple and not rely on orchestration

The main differences between the models at a practical level are non-functional: scalability versus simplicity, simultaneous atomicity versus a synchronisation model, and the extent to which code may need to be deployed more than once.

At a functional level, the project demonstrated that the more federated architecture could meet all stated functional requirements, including locking, the ability to tokenise assets, programmability, DvP, and in a way that met the project's stated privacy requirements.

End-to-End Considerations

Atomicity and Transactionality

When all assets are on the same Tokenisation Platform it is possible, under some models where a single shared ledger is used, to have a single consensus provider (such as a single notary) for all partitions, potentially enabling atomicity, as discussed above.

However, in the general case where assets live in different underlying ledgers, including potentially ledgers based on non-DLT technologies, a question arises as to who is responsible for managing the settlement process between banks when payments between their customers takes place.

Two options were considered:

- Provide APIs for developers to call to invoke the relevant settlement transactions after making a retail transaction. We rejected this approach as it doubles the work required by users for each transaction and is a fragile model with a high chance of error. However, it might create flexibility if or when net settlement scenarios are required.
- Provide APIs which automatically perform settlement transactions when they are called. In a production system, failure of the settlement transaction to complete would cause a rollback of the retail transaction. Note that the failure/rollback scenarios were not in scope for this project.

For the Experimentation Phase, two models were explored for the design in addition to the model described above for the tokenised deposit settlement scenario: one for high value payments and one for low-value payments. For high-value payments, a model where the interbank settlement asset (that is, central bank money) is locked was explored. To maximise the value of the experimentation by exploring multiple options, a settlement model based on scheme rules without underlying technical support was considered for low value payments.

For the high-value scenario that was demonstrated, the system uses a set of hash timelocks for the assets being moved, with the same secret used in each lock, in order to *assure* settlement. On other words, the design was intended to ensure that *if* one leg occurred then the beneficiary of the second leg would *assuredly* be able to receive their expected asset or payment.

An asset to be moved is locked with the hash of a secret, and the orchestration layer uses the hash to place locks on the other assets to be moved – such as commercial bank funds (tokenised deposit on the tokenisation platform), and the central bank funds for settlement – either on the tokenisation platform's wCBDC partition or directly on the RTGS using the synchronisation API.

Note that the Bank of England Synchronisation API is not published, and we have simulated the functionality using hash timelocks in the orchestration layer[59]. However, it could equally be implemented with a third-party lock functionality and a trusted 'synchronisation operator'. The functionality and demonstration would not be affected if it were implemented in this way.

If the asset is claimed, the secret is revealed, providing the information needed for to enable the transfer of the other locked funds. If the asset is not claimed, the hash timelock will expire, freeing up the funds.

59 This is an example of providing consistent functionality across forms of money and across ledger types in a way that 'upgrades' the capabilities of one or more systems rather than 'levelling down' to a low-function lowest common denominator.

Special case for low-value payments

Low-value payments have less value at risk and are often netted off. While some low value real-time payment systems do settle before clearing (e.g. TIPS, TCH) it is uncommon for real-time retail payments systems, and most operate on a deferred net settlement basis. While the Experimentation Platform can operate in both modes via workflow configuration, we demonstrate the more common case.

In this scenario, the commercial bank money is transferred first, initially by a burn of tokenised deposit funds on the part of the originating bank. The orchestration layer will request this when the originating bank calls the API to make a transfer. The API called is cryptographically signed and the end-to-end ID for the transaction is provided by the API and provided to the Tokenisation Platform.

The beneficiary bank will receive a notification message from the platform via the API advising it of incoming funds, again, including the end-to-end ID and ISO20022 data provided by the originating bank (for the beneficiary bank to evaluate the transaction). The originating bank must then validate the Tokenisation Platform transaction. If it declines to do so, the transaction is aborted, and the API will return a failure code.

Assuming the burn is completed successfully, the orchestration layer will request a mint of funds on the part of the beneficiary bank, again referencing the transaction by its end-to-end ID. The beneficiary bank will evaluate the transaction and validate or decline the transaction. If the transaction is declined, the orchestration layer will inform the originating bank so that it can re-mint the burned funds. This is a negative path and is not technically demonstrated.

If the transaction is successful, the orchestration layer will then request the move of central bank funds between the two banks. In the Experimentation Platform we perform a settlement transaction for every commercial bank money transaction, but it is at this point that a production system could carry out netting off, which is the key benefit or rationale for considering this approach. In a later phase, negative scenarios would need to be modelled, to ensure that the impact of payment rejections, reversals and timeouts was adequately handled.

Transaction Reversal

A related concept to transaction atomicity and coordination is transaction reversal. This refers to the set of operations required if there is a requirement to 'undo' a previously confirmed transaction or set of transactions.

The approach taken for the Experimentation Phase was to adopt a 'fix forward' model. In other words, given the complexity inherent in managing transactions across multiple ledgers and assets and asset issuers, the approach adopted was one where the solution to, say, a mistaken transaction was for a new transaction to be performed, with the net effect that the two transactions cancel each other out.

One consequence of this is that consideration would need to be given to the parties who should have the ability to instruct such transactions and the extent to which the underlying ledgers, or permissioning system on the API and Orchestration Layer, should have special case logic to override the need for approvals and signatures from all parties in the case that an instruction is issued by a suitably authorised party.

Synchronisation of Records

A further complication arises under some tokenisation models if the API and Orchestration layer is responsible for coordinating the update of records held within banks to track activity on a Tokenisation Platform

When a bank chooses to maintain a pooled or mirror account model to manage tokenisation this is relatively straightforward, as the settlement accounts cover many retail accounts and can be held as reference data in the orchestration layer.

For Shadow accounts held on a one-to-one basis, it is more complicated as a different account is needed for each transaction. Without having to enrol each retail customer, this information has to be provided by the bank initiating the transaction and it needs to know the beneficiary account and the beneficiary shadow account, potentially complicating the API.