ONYX by J.P.Morgan

digital currency initiative

mit media lab

Application of Programmability to Commercial Banking and Payments

Acknowledgements

Authors

Wee Kee, Toh
Global Head of Business Architecture for Coin Systems | Onyx by J.P. Morgan

Anders Brownworth
Senior Research Advisor | MIT Digital Currency Initiative

Robert Bench
Senior Advisor | MIT Digital Currency Initiative

Nicole Li
Student Researcher | MIT Digital Currency Initiative

Sazma Sarwar
Student Researcher | MIT Digital Currency Initiative

Contributors

Heiko Nix
Siemens

Kelvin Li
Ant International

Umar Farooq
J.P. Morgan

Naveen Mallela
Onyx by J.P. Morgan

Shekhar Gahlot
Onyx by J.P. Morgan

Muh Hwa, Lee
Onyx by J.P. Morgan

Sai Valiveti
Onyx by J.P. Morgan

Daniel Chew
Onyx by J.P. Morgan

Contents

Chapter —— 01

Executive Summary

This paper provides an overview of blockchain technology and smart contracts and explains how they enable multiple parties to deploy and execute code in a trusted manner on a common platform. It introduces the concepts of bank-side programmability and how it is enabled by allowing clients to deploy their logic, in the form of programmable instructions, in the bank's environment. Through contrasting bank-side programmability with client-side programmability, the paper examines potential benefits and trade-offs, and highlights the design considerations of implementing programmability in payments.

Produced by a joint research team composed of staff from the Massachusetts Institute of Technology Digital Currency Initiative (MIT DCI) and Onyx Coin Systems at J.P. Morgan, this paper draws from the experience of MIT DCI in digital currency research and programmability to describe and explain programmability in the context of banking and payments. It draws additionally on real-world examples of how Onyx is applying programmability to solve clients' needs to further elaborate on how programmability can be applied, and the expected benefits from doing so.

The paper will proceed to describe the application of programmability specifically to the area of commercial banking and payments. This will be achieved by using the examples in the context of a specific platform, namely JPM Coin and Programmable Payments, which are live products of Onyx by J.P. Morgan. The research team selected these existing products as examples of a digital currency platform that a financial institution could use to offer corporate customers the ability to automate some financial activities. Instruments offered by other providers may be suitable for programmability as well. References to JPM Coin or other offerings and services of Onyx, J.P. Morgan Chase & Co., or of other providers do not imply that MIT DCI has endorsed these products or services.

The use of live products and systems provides a real-world overlay on complexities of implementing programmability for banking and payments, highlighting not just technical considerations, but also business, governance and operational considerations in designing and implementing programmability capabilities for banking and payments.

The paper will then explore three potential use-cases relating to:

(i) dynamic funding in treasury management,
(ii) automated release of payments through third-party logistics providers, and
(iii) automated margin funding and settlement.

Through examples of real-world scenarios that clients face today, the paper describes and explains the opportunities, how programmability can be applied, and the expected benefits that programmability will bring. The paper also highlights forward-looking possibilities where programmability can be used to further improve the solutions in ways that are not viable with conventional architectures today.

The final section describes key areas that call for further exploration, including better means of managing concerns of applying programmability to banking and payments, and sets the basis of a research agenda for future exploration into this area.

Chapter ——— 02

Introduction

The growth of programmability on public blockchain networks has spawned a thread of research on the application of programmability to real-world use cases. Computer scientists and technologists are building programmability onto distributed ledgers to solve old problems and create new opportunities.

Programmability is the ability of a system to execute specific tasks based on provided instructions.

It is not a new concept - the first programming language was developed in 1843 [1]. Modern programming languages emerged in the 1940s, as sets of low-level instructions closely tied to the hardware on which they ran. Further evolutions into higher-level programming languages occurred in the 1970s, where engineers developed more abstraction to hide implementation details. Modern programming languages continue to evolve, making it easier to write code, and in the process, making programming accessible to more people. Recent developments in Low Code / No Code (LC/NC) development tools, along with advances in artificial intelligence platforms, have further democratized software development by making it easier to create applications without the need for users to know how to write code.

[1] Ada Lovelace is credited with the first programming language, to support Charles Babbage's Analytical Engine, a prototype to the modern programmable computer.

The ability to make use of functions developed by others has made it even easier to develop applications. Software libraries, which contain code and functions previously developed by others, can be reused easily, simplifying development time. Smart contracts on blockchain networks take this to an even higher level, allowing applications in the form of smart contracts to easily interact with other smart contracts, in what is termed composability. This ability of applications interacting directly with other applications provided by others could improve interoperability and lead to the provisioning of a richer suite of products and services.

The combination of easier development and deployment of software code in the form of programmable instructions, and the ability for these instructions to interact directly with each other, is creating new ways of enabling programmability, and making it more accessible to a greater number of users. In this paper we explore the application of programmability to commercial banking and payments use cases, with a focus on improving efficiency and enabling new use cases and opportunities.

Chapter — 03

Programmability on Blockchain Networks

Programmability is the ability of a system or database to execute a specific task on instructions provided. On modern computers, one can write instructions in various languages and the computer provides an execution environment to run the code[2].

The most well-known smart contract platform is Ethereum. Ethereum is a public, permissionless blockchain capable of executing generic smart contracts using the Ethereum Virtual Machine (EVM). Nodes in the Ethereum network use a proof-of-stake consensus mechanism to agree upon the current state of the blockchain. New transactions entering the system are processed by each node leading to a newly proposed state of the blockchain. The Ethereum ecosystem presents a system where multiple parties can deploy smart contracts on a common platform, and where these smart contracts can interact directly with each other.

[2] George, Nikhil, Thaddeus Dryja, and Neha Narula. “A Framework for Programmability in Digital Currency” arXiv preprint arXiv:2311.04874 (2023)

The trusted execution of code deployed by different parties on a common platform enables a paradigm shift in designing how systems interact. A common platform with adequate safeguards can be made to execute code from more than one external source. This makes data sharing and compositions made of other externally contributed functions relatively simple.

In conventional system design, it is uncommon for a transaction processor to also process arbitrary code. As an example, a payment system is usually designed for a primary purpose of processing payments. It will make available specific functions, such as initiating a payment, which it will process. Any logic leading up to the initiation of a payment is processed externally, and only the request to initiate a payment is transmitted to, and processed by, the system. As we'll discuss, enabling the transaction processor to execute code not strictly associated with account balance updates could provide additional benefits.

The ability for a transaction processor to also process arbitrary code enables a paradigm shift in programmability, and will be explained in the next section in the context of client-side and bank-side programmability.

Chapter ——— 04

Client-side and Bank-side Programmability

In the context of banks and their clients, clients have had the ability to implement specific logical instructions in the form of software code, which then interacts with their bank's systems for banking services, such as initiating a payment. The code is typically deployed and executed within a client's environment, such as their own applications, and then interacts with the bank's systems through communication channels such as APIs[3]. We refer to this architecture as "client-side programmability".

[3] While the paper generally refers to communications via Application Programming Interfaces (APIs), clients have other options of communication channels, such as through the SWIFT messaging network.

The specific innovation of “programmability” or “programmable payments” that is being referred to in the paper, is the ability for clients to deploy their logic in the bank’s system, with the transaction processor, in the form of programmable instructions. In this manner, the programmable instructions are deployed as self-contained executable instructions with the bank, and can be executed directly by the banks’ systems without further dependency on the clients’ systems. We refer to this as “bank-side programmability”.

The differences of “client-side” and “bank-side” programmability are explained through a simplified example of implementing a target balance.

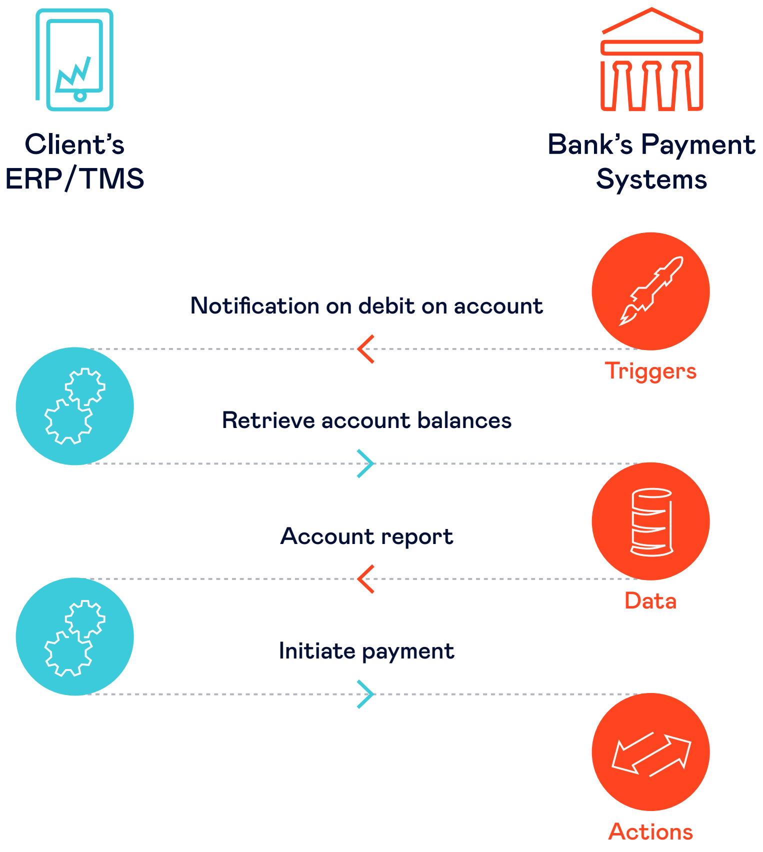
Implementing target balance as client-side or bank-side programmability

Target balance is a common treasury management technique to maintain a pre-specified balance in a bank account. It involves gathering up-to-date information on the balance of an account, and initiating payment transactions at pre-specified times to move funds into or out of the account to achieve the pre-specified target balance. Examples of how target balance can be implemented through (i) client-side programmability and (ii) bank-side programmability is described below, to better understand the differences between the two models.

Target balance through client-side programmability

A conventional model of implementing target balance through client-side programmability has the rules or logic deployed and executed in the client’s system, which will communicate with the bank’s system to retrieve account data, and to initiate payments. In the case of payments, communications will typically take place through channels such as APIs or the SWIFT network, using standardized message formats such as SWIFT MT (message text) or XML-based ISO20022 MX. The specific messages are included in the example below as a reference.

The event trigger, such as an account being debited or credited, originates from the bank, and is transmitted to the client in the form of a Debit or Credit Notification (in the message format of MT900/MT910 or camt.054). The client’s systems will process the notification, and request more information as necessary. If it requires additional data such as the account balance, it will request the data through an Account Report Request (MT920 or camt.060), which the bank will then provide through an Intraday Account Report (MT942 or camt.052). The client’s systems will use the data to evaluate the account balance and determine the amount of additional funds needed to maintain the target balance, if any. The action of moving funds will be requested through a Payment Initiation (MT101 or pain.001).



Target balance through bank-side programmability

In bank-side programmability, the client's logic, in the form of programmable instructions, is deployed and executed on the bank's system. As the programmable instructions reside within the bank's system, which has direct access to the account data and the functions of initiating payments, no further information or communication is required from external systems or parties for execution. Back-and-forth communications between the bank and client's environments are eliminated, with all processing completed within the single environment of the bank.

It should be noted target balancing is often already offered by banks as cash management products, with instructions executed within the bank's environment. Such products offer limited options for customization, and lack the code or logic expressivity of programmability solutions. A conventional target balance set-up is typically time-based (i.e. triggered at a fixed time), with limited options for conditions (i.e. above a pre-specified amount) and actions (i.e. move excess funds to a pre-specified account). Programmability significantly expands the scope of customization, by allowing for event based triggers (i.e. debit on account), conditions (i.e. complex logic as allowed by expressivity of the code), and actions (i.e. move funds from multiple accounts).

Benefits and trade-offs of bank-side programmability

This new model of bank-side programmability creates new possibilities by allowing clients to deploy their logic in the form of programmable instructions in a bank's environment. It is expected that bank-side programmability will provide benefits over client-side programmability on conventional systems, such as:

01 Improve execution and response time as the triggers, logic and actions are all performed within a single environment. This minimizes the "technical" lag of communicating across platforms, and the "business" lag where updates are typically pushed in batches or polling is limited in frequency.

02 Improve certainty of execution and minimize failures, as instructions are housed and performed within a single environment, and there is no dependency on clients' systems.

03 Improve traceability and auditability, as execution is performed in a single environment and data is maintained in a common format.

04 Broaden the range of programmability by granting access to data and event triggers that were typically not available or transmitted to clients, and allowing for programmable instructions to be processed within a payment, rather than after a payment. This is now possible due to the improved execution time and certainty of programmable instructions, which allow them to be integrated into a payments flow, without significant negative impact on the overall processing time and certainty.

05 Enable payment instructions to be performed close to, or even after typical bank cut-off time, as compared to conventional channels. This is possible as programmable instructions are executed with higher certainty and lower possibility of failures, and hence require lower time buffers and no additional input from other parties.

06 Enable interactions of programmable instructions or composability, which could support more complex use-cases. Atomic operations of such linked instructions ensure that they either completely succeed or completely fail. This is particularly useful for linked obligations, such as a Delivery-versus-Payment or Payment-versus-Payment transactions where it is important that the linked obligations are discharged either altogether or not at all, to minimize counterparty risks during settlement. Eliminating partially completed transactions ensures transaction and state consistency, which reduces system complexity, need for reconciliation, and the need for manual recovery processes.

There are trade-offs, where conventional systems might continue to have an advantage, or where bank-side programmability could face limitations:

01 Conventional systems maintain a clearer separation of responsibility. Interactions between client and bank in conventional architecture are clearly delineated when they communicate through APIs requests and responses, and are also well captured in audit trails on both sides. This can be improved by designing audit capabilities that clearly capture the interaction of the programmable instructions by the client, and the resultant transaction processing by the bank.

02 Traditional systems lend themselves well to highly compute-intensive, real-time operations. Smart contract operations by comparison are computationally limited and may not complete in well-defined timeframes. While the need for compute-intensive or real-time decision making is generally limited, especially in treasury operations, tasks such as high frequency trading may be better suited to the traditional model.

03 The consistency of processing is dependent on consistency of external data. The blockchain network provides coherence guarantees with on-chain processing, but not with off-chain processing and data. To use external data, blockchain networks implement oracles. Oracles are data feeds that make off-chain data sources available to blockchains for smart contracts[4]. Importantly, oracles can be used to improve assurance on the transmission of data onto the blockchain for processing. It is worthwhile to note that concerns on external data are not specific to blockchain networks, but remain a concern for centralized systems.

04 While the high degree of code expressivity allows users to develop complex logic, this ability to write, deploy and execute arbitrary code also raises governance concerns and cybersecurity risks. The execution of poorly written code could result in degradation of service; malicious code

[4] https://ethereum.org/en/developers/docs/oracles/

could be offered that is intended to cause harm or financial losses. One broad option of managing such risks is through deliberate restrictions on expressivity, but this also limits the possibilities of what could be achieved with programmability. Various possible mitigation techniques are further described in the section on future exploration.

05 The deployment of client logic with the bank could be considered an expansion of the roles and responsibilities of the bank's systems. Besides governance considerations of the type of logic that can be deployed and the use-cases that can be supported, technical considerations exist regarding the scalability and performance of deploying multiple applications and use-cases on a shared ledger platform.

Chapter —— 05

Overview of JPM Coin and Programmable Payments

We turn now to consider the application of programmability in commercial banking and payments using as an example the context of a specific and existing platform, namely JPM Coin and Programmable Payments, which are live products of Onyx by J.P. Morgan. These are examples of platforms that a financial institution could use to offer corporate customers the ability to automate some financial activities. Instruments offered by other providers may be suitable for programmability as well.

JPM Coin is a blockchain-based demand deposit product offered by J.P. Morgan that allows their clients to make cross-border payments between JPM Coin accounts on a 24x7, real-time basis, with most transfers completing in seconds. Customer demand deposit accounts are recorded on a private, permissioned blockchain ledger, and are referred to as Blockchain Deposit Accounts (BDAs). The blockchain ledger in which BDAs are recorded represent the official record and evidence of J.P. Morgan's deposit liability owed to each participating client under current applicable U.S. banking laws and regulations. Consequently, J.P. Morgan records a deposit liability on its balance sheet representing a balance in a BDA in the same way it records a liability representing a balance in a traditional demand deposit account that is recorded using non-blockchain technology.

JPM Coin is a live deposit product, with daily transaction values exceeding USD $1 billion. It also provides additional capabilities that are not typically provided by conventional payment systems, including programmable payments where clients can define programmable instructions that are deployed and executed within the J.P. Morgan environment.

JPM Coin

JPM Coin is built on a combination of blockchain-based and non-blockchain-based components, which are collectively referred to as the JPM Coin platform.
The blockchain components are built on Quorum, an (EVM)-compatible blockchain platform that conforms to the Enterprise Ethereum specification maintained by the Enterprise Ethereum Alliance. While the blockchain is currently deployed as a single-bank platform with all blockchain nodes controlled by J.P. Morgan and hosted within the bank's environment, it is designed with a future view towards externalizing of nodes, where clients will be able to host nodes and connect directly to the JPM Coin platform[5].
The underlying technology has also been deployed in Partior as a multi-bank shared ledger, where participating banks host their own nodes [6].

The ledger component of JPM Coin, which records the balances of the clients' accounts, is on-chain - meaning the balances and updates to the balances are recorded and performed on the blockchain. The payments processor component of JPM Coin orchestrates the payments processes, including connecting with other J.P. Morgan systems in order to complete other processes like fraud checks, sanctions screening, and other payment control and compliance processes. Due to the high-speed synchronous nature of payment orchestration, and the amount of data processed, the component is primarily off-chain — meaning the processing is performed outside of the blockchain. State changes (the change of the transaction state) is recorded on-chain, which allows for the possibility of state changes triggering other smart contracts. This can also be viewed as the composability of smart contracts.

[5] Externalized nodes is not currently offered. Such an offering is subject to J.P. Morgan obtaining all required internal approvals and regulatory approvals and completing legal, compliance, risk and other due diligence.

[6] The technology underpinning JPM Coin is also licensed to Partior. Partior was established in 2021 as a joint venture between J.P. Morgan, DBS Bank and Temasek, with Standard Chartered joining in 2022. Partior brings multiple institutions on a common network to enable cross-border payments and foreign currency transactions to be conducted with a common set of protocols to improve efficiency.

An example of client-side programmability is included to explain how a client typically interacts with JPM Coin for payments. This case study reflects the independent work of a J.P. Morgan client and particularly interesting as the client's logic is also deployed as smart contracts on its blockchain platform. While the client's application currently resides on a different blockchain platform and communicates with the JPM Coin platform via APIs, there is a possibility for composability with JPM Coin's smart contracts in the future.

Technical options that could enable composability, and implications are discussed in a subsequent section on future exploration.

Use Case:

Real-time, 24x7 Treasury Management using JPM Coin

Ant International, a leading global digital payment and financial technology provider, supports merchants of all sizes worldwide to realize their growth aspirations through a comprehensive range of tech-driven digital payment and financial services solutions, in close collaboration with partners. Due to the global nature of ecommerce transactions, it initiates and receives payments around-the-clock in multiple locations. Partnering with more than 70 global financial institutions worldwide, it provides online payment channels for 1.2 billion buyers and 2 million sellers in over 200 countries, servicing major global merchants and all Alibaba affiliates.

With an objective of improving treasury management processes, it developed an internal blockchain platform to record account balances across its global entities. The blockchain platform is integrated with its Treasury Management System (TMS), which is used to forecast and manage liquidity. Ant International's TMS is powered by its proprietary AI-based programmable triggers, which are integrated with the blockchain platform.

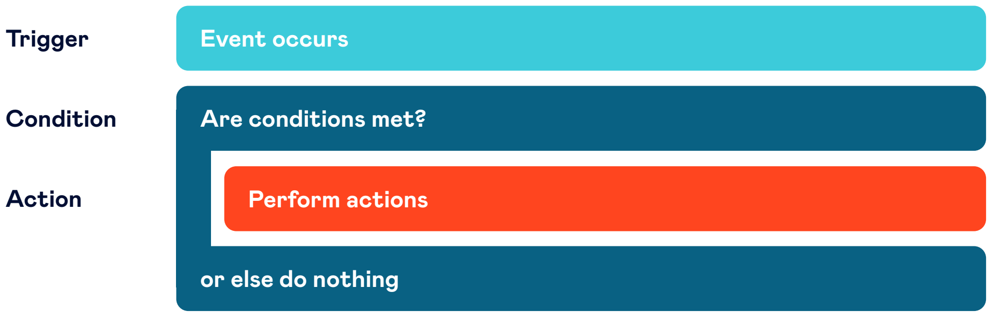
The algorithm automates real-time monitoring of the liquidity usage across financial hubs and forecasts upcoming business needs, initiating payment instructions based on accurate, live events. For example, funds in excess of forecasted needs could be transferred to a consolidation account where they can be deployed to higher yielding instruments. Another example is in moving funds ahead of a forecasted business need, to ensure that the account has sufficient funds to make payments. Payment instructions initiated via its blockchain platform are executed on the JPM Coin platform to move funds cross-border and cross-entity in real-time across key hubs. While the payment instructions are initiated by smart contracts on Ant's blockchain platform, and processed by smart contracts on the JPM Coin platform, the smart contracts reside on different blockchain platforms, and currently communicate via APIs. The implications of composability of the smart contracts are discussed in a subsequent section.

API connectivity between Ant International's blockchain platform and JPM Coin enables seamless movement of funds and real-time synchronization of balances, ensuring data consistency without the need for reconciliation. A customized balance synchronization methodology ensures constant consensus of information state maintained on the two permissioned blockchain networks, avoiding any additional reconciliation requirements.

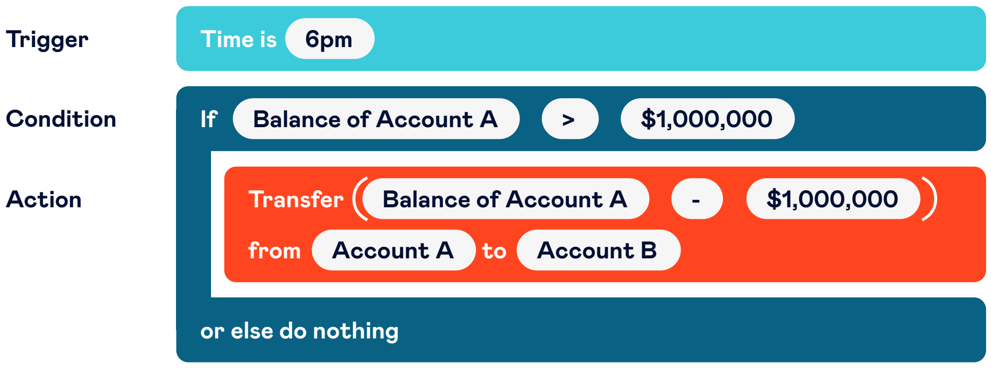
The fully integrated treasury management solution has enabled seamless, 24x7, cross-border settlement across its global treasuries, covering USD across multiple entities and many accounts. It has also resulted in business benefits, including additional business flows, reduced funding costs, and reduced liquidity risks. JPM Coin has processed billions of dollars seamlessly with Ant International since inception.

Programmable Payments on JPM Coin

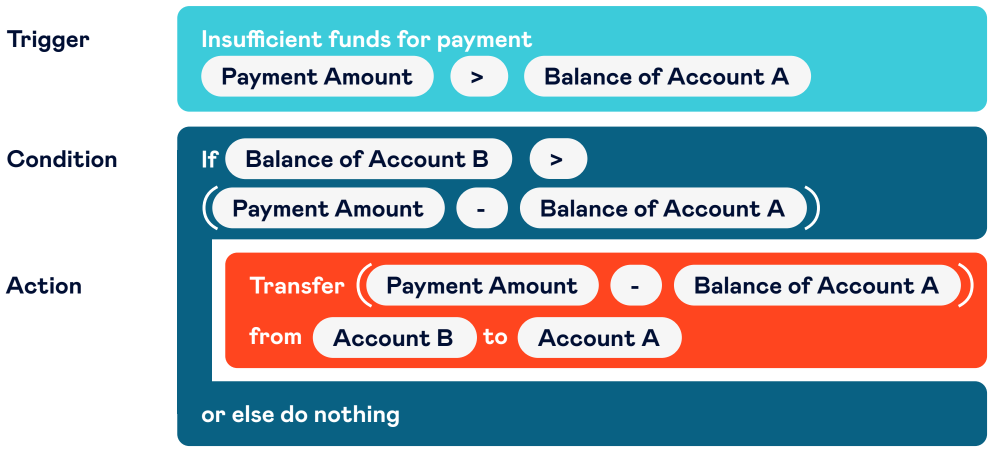
Programmable payments is a capability offered by J.P. Morgan that allows clients to define programmable instructions, such as a set of event triggers, conditions, and actions, that are executed on the JPM Coin platform. This can be expressed as: when an event occurs, evaluate if conditions are met, and perform actions.



One example is a programmable instruction to transfer funds in excess of a pre-specified amount at the end of the day. In this example, there is a time-base trigger which is activated at 18h00 hours (defined as "end of day" in local time), to check the condition of whether the account balance exceeds pre-specified amount, and if this condition is met, to perform the action of moving excess funds to a pre-specified account. This also includes logic to calculate the excess funds to be transferred, through an arithmetic operation of account balance minus pre-specified amount.


Trigger
Condition
Action
Time is 6pm
If Balance of Account A > $1,000,000
Transfer (Balance of Account A - $1,000,000)
from Account A to Account B
or else do nothing

A second example is the need to draw funds in from another account when there is insufficient funds to complete a payment. Here, there is an event-based trigger which is activated when there are insufficient funds to complete a payment, and performs an action of making a transfer from another account to this account. In this example, the event is triggered while processing a payment, resulting in a pausing of the payment being processed, which is resumed when the programmable instruction is completed.


Trigger
Insufficient funds for payment
Payment Amount > Balance of Account A
Condition
If Balance of Account B > (Payment Amount - Balance of Account A)
Action
Transfer (Payment Amount - Balance of Account A)
from Account B to Account A
or else do nothing

The range and complexity of programmable instructions that clients can build and deploy is determined by (i) the expressivity of the instructions, (ii) availability of data, including event triggers; and (iii) access to functions that perform different actions. For example, restricting expressivity, such as disallowing the use of loops and recursion, will limit programmable instructions to linear processing, and may limit the complexity of logic that can be implemented. A greater availability of data, including data that was not previously available to clients, such as interim transaction states, can further expand the range of programmable instructions that clients can develop. Greater access to functions and actions has a similar effect. Besides functions provided by the bank, there could also be services made available by other parties[7] on the platform in the future, such as foreign currency exchange through a third-party market maker. Such services may be provided as components or functions, which can be composed by clients into larger services with more features. This leads to an ecosystem of network effects expanding the options for users of the platform.

The JPM Coin platform is designed with an architecture that enables access to a much broader range of functions, data, and event triggers for programmable payments. Instead of a workflow-based design typically used by conventional payments processing systems, the platform is designed around states and transitions.

A typical workflow-based design has a single fixed pathway for each payment flow, with alternative pathways often limited to exception handling. Events are generated when workflow processing is complete, which typically only returns a success or failure status. Even if events are generated for interim statuses as part of the processing, the processing is not designed to be interrupted. Events of interim statuses can only be used for notifications, they are not able to change the processing.

[7] Support for third-party services on JPM Coin is not currently offered. Such an offering is subject to J.P. Morgan obtaining all required internal approvals and regulatory approvals and completing legal, compliance, risk and other due diligence.

The state-based design used by JPM Coin defines a number of states for each payment flow, with the possibility of multiple pathways to transition between states, enabling a highly agile and flexible design. State transitions are accompanied with various cues, also called events, which are used to trigger other actions, including actions to transition to the next state. A typical workflow can be remodeled as a set of states, and bank-defined actions to transition between these states. In addition, it could also include client-defined programmable instructions, or actions that are triggered by the state change events, to perform specific tasks.

This allows programmable instructions to support use cases like the above example of dynamic funding. In a typical payments workflow, insufficient funds in the debiting account will likely result in a failed payment. With programmable payments and JPM Coin, clients can deploy programmable instructions that are executed prior to the bank-defined step of checking for available funds. This could range from a simple action of drawing funds from a pre-specified account such as in the above example, to more complex rules that determine which account to draw from based on relative balances and time of day.
As further actions are made available to clients, the programmable instructions could even perform FX to fund the debit account. In essence, funding of the debit account is performed just-in-time and clients' payments are performed without the need to pre-fund the debit accounts.

Chapter ——— 06

Application of Bank Programmability in JPM Coin

The programmable payments capability offered by J.P. Morgan, together with JPM Coin, can be applied by J.P. Morgan clients in different ways for different use cases.Three client use cases are detailed to describe and explain how bank-side programmability can be applied to corporate payments. The use cases include immediate applications as well as future-state use cases that require further enhancements of the business applications and the payments processing. By showing the art of the possible, these use cases aim to advance understanding of programmability, and promote further exploration within the financial services industry. For each use case, we provide background, pose an opportunity for improvement, a solution that employs programmability, further enhancements building on the proposed solution, and a discussion of the benefits offered through programmability for that example. As with the prior case study, the following case studies reflect the independent work of J.P. Morgan clients.

Use Case: A

Treasury Management — From Forecasting to Dynamic Funding

Programmable payments that are deployed within a bank's environment enable execution of payments in real-time and with a high degree of certainty. Coupled with the ability to fund outgoing payments with other accounts in real-time, a typical treasury management objective of maintaining a target balance can be achieved without the need for forecasting, which is probabilistic, uncertain, and often inaccurate. This section describes how Siemens AG and Onyx by J.P. Morgan are exploring the application of bank-side programmability to treasury management, and how this allows moving away from a model of forecasting to dynamic funding.

01 Background

Large multinational companies (MNC), such as Siemens, typically hold accounts in different countries and in different currencies. These accounts need to be maintained for financial activities that take place locally, such as making payments to local suppliers and collecting payments from local customers. Transactions on the accounts can be broadly categorized as debits which are transactions that reduce the balances, i.e. outgoing payments, and credits which are transactions that increase the balances, i.e. incoming collections. If at any point, debits exceed balances, outgoing payments may fail due to insufficient liquidity, or the account may go into overdraft. Excess balances at the end of the day incur opportunity costs, as they could have been deployed to other instruments for higher yields.

With an objective of maximizing returns, MNCs typically have treasury management functions that consolidate surplus funds across the accounts, and place them in higher yielding instruments or accounts. Surplus funds are balances in excess of the working capital required for making payments in the course of normal operations. This objective typically translates to a target balance for each of the accounts, and a goal of keeping account balance as close to the target as possible at the end of every day.

Forecasting is a means to minimize the variance of actual end-of-day balances from the target balance, by projecting or estimating the debits and credits for the day to determine the optimum working capital to be maintained in the account, and the expected surplus funds to be moved out. This can be done with data on invoices and due dates, which gives an indication of when customers will make a payment, and when they will make a payment out.

Forecasting, however, is probabilistic and is often not accurate. The accuracy of cash-forecasting decreases with a tightening time gap or within a more precise time frame. As an example, an invoice will indicate a high probability of incoming funds on or near to the due date (a range of days), but the probability of receiving funds on a specific date such as the due date itself would be lower, with some payments being received a day earlier or later. This could be due to uncertainty in the processes of the payer, and exacerbated by the cross-border payments process of going through multiple intermediary banks.

Inaccurate forecasting can lead to payment failures and higher costs. For example, an expected incoming payment that is received a day later could lead to insufficient funds for outgoing payments, resulting in payment failures, or an account going into overdraft. An unexpected incoming payment could result in surplus funds that were not deployed in the most capital-productive manner.

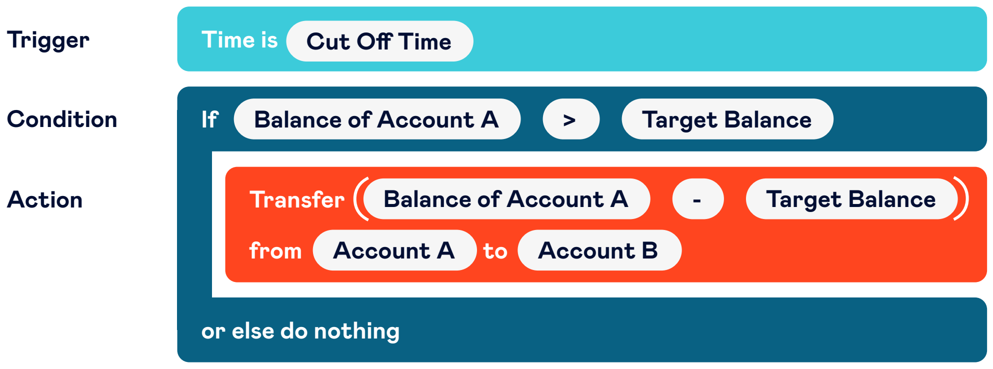
02 Opportunity

Programmable payments can enable target-balancing and the broader objective of minimizing fees and maximizing yields, in a highly certain manner, without the need for short term forecasting. As the programmable instruction is executed within the bank's environment, it has access to all required data, and can complete processing quickly and with high certainty. The instruction to move funds out can be executed at close to cut-off time, ensuring there is no excess balance and eliminating the opportunity costs of that. This leaves a remaining concern of payment failures if there are insufficient funds. This can be addressed by deploying programmable instructions that draw funds from other accounts if there are insufficient balances. This is possible as transfers across clients' accounts through JPM Coin can take place in real-time, 24x7, so a client can use funds available in any of their other accounts to fund a payment.

03 Solution

The client can deploy two sets of programmable instructions. The first "Zero Balance" instruction activates at the pre-specified cut off time, i.e. 6pm, and moves all excess funds to the designated account. The second "Dynamic Funding" instruction activates whenever there is a payment that will fail because of insufficient funds in the account. The instruction will fund the account with funds from the designated account. Once funded, the original payment will continue processing as there is now sufficient funds in the account.

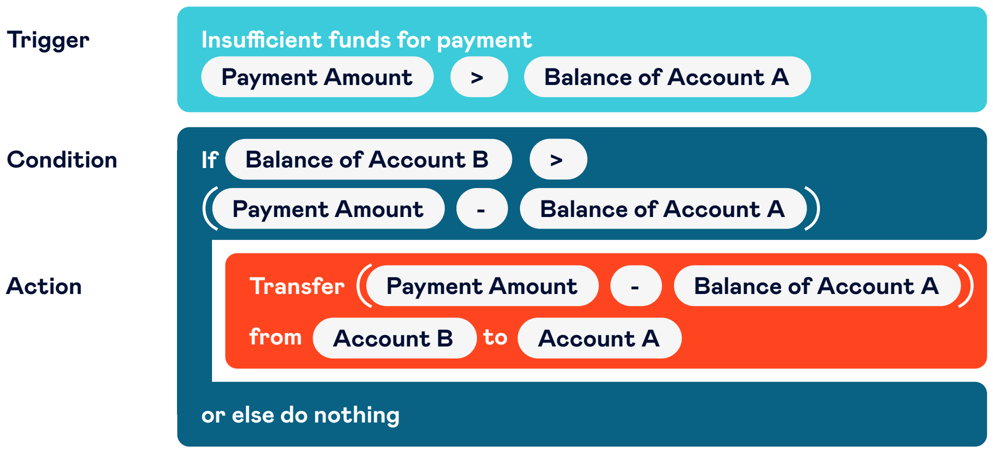
Zero Balance



Notes

Cut Off Time	Is set at 6pm
Target Balance	is set at $0
Account B	Is set at the designated account all the excess funds are transferred to

Dynamic Funding



Notes

Account B	Is set at the designated account to draw funds from

04 Possible enhancements

While the example showed the drawing of funds from a single account, it is possible to set up more complex logic, such as drawing from a cascading set of accounts until there is sufficient funds, or performing foreign currency exchange (FX) to fund the account using balances in other currencies. The logic can also include rules for more fine-grained control, such as setting amount thresholds before the actions are initiated. Clients may also be able to view balances and initiate payments for their accounts with other banks, if there are appropriate APIs made available.

05 Benefits

The client will maximize yield and reduce opportunity costs by moving the available funds to higher yielding accounts or instruments every day. At the same time, they can minimize unexpected payment failures, or draw on credit lines. These benefits are possible through real-time automated execution of payment instructions within the bank's environment.

The fine-grained logic also allows for better control over what actions and transactions will be performed, and improve overall cost efficiency. This is as the transaction incurs costs, not just from a payment processing perspective, but also from the operational efforts of monitoring, validating and reconciling transactions. Logic can be put in place to evaluate and perform a transaction only if it is net-beneficial.

While there are other products such as cash sweeping and pooling that can provide a similar solution, programmability places the control in the hands of the clients, allowing them to set up much more complex and fine-grained logic and rules. Clients are able to set up events-driven rules, which are triggered when certain events occur, rather than just static time-based set-ups. Also, clients can set up and amend these rules in a self-service manner at their convenience, which enables a much faster time-to-market.

Use Case: B

Automated release of payments through third-party logistics providers

Trusted data from neutral third-parties, such as a logistics provider or through devices connected to the Internet of Things (IoT), can be used to enable the release of payments, and eliminate frictions in the payment process between the supplier and buyer

01 Background

A typical trade scenario involves a supplier delivering goods to a buyer, with the buyer making payment either upon delivery, or based on a fixed period after delivery and invoicing. While most transactions are uncontentious, the underlying incentives for suppliers and buyers are not aligned when it comes to payments. Suppliers want to receive funds earlier, while buyers would prefer to pay later. Furthermore, there may also be a general lack of trust between the parties, with buyers preferring to receive goods before making payment, and sellers preferring to release goods only when there is certainty of payment.

Disputes may also arise when acceptance criteria are subjective. For example, in the case of temperature-sensitive products such as pharmaceuticals or perishable food, buyers may be unwilling to accept delivery if there are indications that products were subjected to adverse temperature or other conditions outside of the permitted range. Resolving such disputes often requires manual investigation and may sometimes require arbitration, resulting in high operational costs to all parties.

02 Opportunity

Trusted data can be used to eliminate the subjectiveness involved in trade and payment processes, and in doing so, improve the certainty of processing and enable automation of these processes. A critical premise for this is that the data, and the source of the data, is trusted.

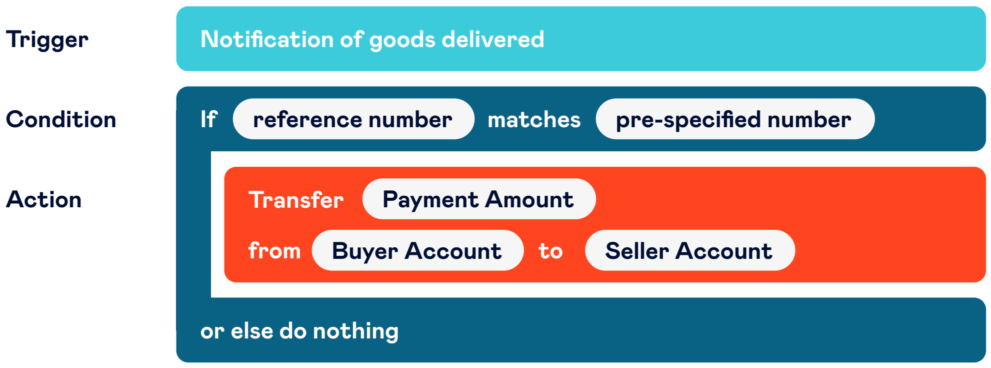
The simplest source of trusted data is a neutral third-party. For example, a third-party logistics provider can provide an objective confirmation of delivery, which is then used as a proof to automatically trigger the release payments. IoT devices can also be used to provide GPS location data that can be used as an indicator that goods have been delivered. They could also monitor environmental data such as temperature and vibration, which could be used to evaluate if conditions exceeded allowed range.

In addition to simple payments, funds could be placed in escrow, and released only when conditions are met, providing assurance to both suppliers and buyers. As the release conditions are objective, the need for manual arbitration may be lower, and hence also lower the costs of providing such escrow services.

03 Solution

The client can deploy a programmable instruction to release payment when a delivery notification matching the reference number is received. When a third party logistics provider delivers the goods and marks the delivery as complete, a notification will be transmitted to the JPM Coin platform. The programmable instruction will act on the notification, which will contain information such as a reference number, it will match the reference number, and will make a payment to the supplier.

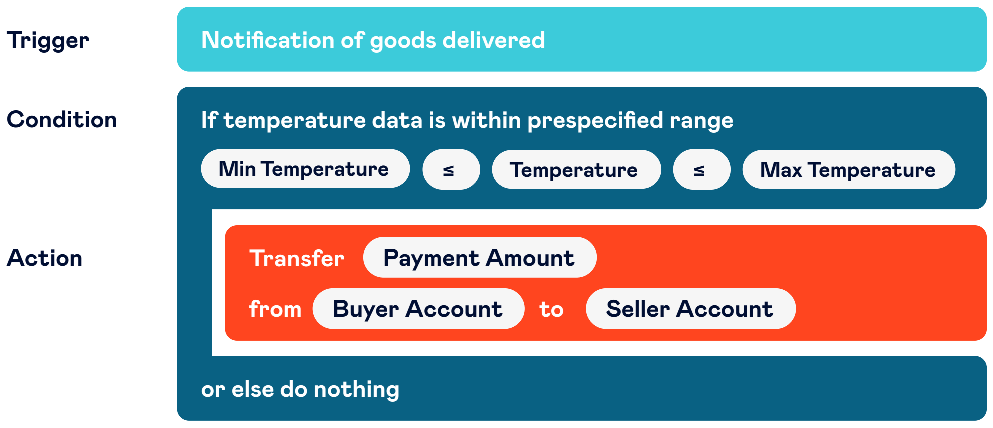
Payment upon delivery notification


Trigger
Notification of goods delivered
Condition
If reference number matches pre-specified number
Action
Transfer Payment Amount
from Buyer Account to Seller Account
or else do nothing

04 Possible enhancements

In addition to the above use-case, the supplier can include an agreed IoT device with the shipment. Upon delivery, the IoT device will transmit temperature data to the JPM Coin platform. A set of mathematical functions can be applied, i.e. to calculate the mean and variance of temperature fluctuations, which is then evaluated against the pre-specified value. Payment is released only if the fluctuations are within the allowed range in the acceptance criteria. If fluctuations exceed range, funds will be withheld pending further manual resolution. It is also possible that over time, resolution can be codified as rules within the programmable instructions, i.e. withholding of different amounts as penalties based on the range of fluctuations.

Payment upon delivery and fulfillment of acceptance criteria


Trigger
Notification of goods delivered
Condition
If temperature data is within prespecified range
Min Temperature ≤ Temperature ≤ Max Temperature
Action
Transfer Payment Amount
from Buyer Account to Seller Account
or else do nothing

The above two programmable instructions are made as payment instructions, and direct a payment from the buyer's account to the supplier's account. It is also possible that funds are managed in a way that provides the supplier with comfort and certainty that funds are available and earmarked for their goods. This could be achieved through a conditional reservation of funds where funds continue to reside in the account, but is earmarked and cannot be used for any other purposes except for release to the supplier when conditions are met. This achieves a similar effect to the use of escrow services where funds are transferred to an escrow account administered by a third party, but with the added benefits that the funds remain in the account and continue to accrue interest, and that there is much lower operational effort, and hence costs, to administering the release of funds.

05 Benefits

This will reduce operational costs and reduce the need for dispute management between the transacting parties. These benefits are possible through cross-organization automation, as both parties trust the execution of the prespecified rules, and the data that drives the execution.

Use Case: C

Automated margin funding and settlement

Funding of margin calls can be automated, reducing the potential for funding failures and forced liquidation of positions for clearing members, as well as risks for the clearing house. Furthermore, automation can reduce the processing time for margin funding, allowing for the potential to move towards more frequent margin funding, which may reduce risks for the clearing house, leading to lower margin requirements and hence improved liquidity for clearing members.

01 Background

In trading of some financial instruments such as futures, traders (also referred to as the clearing members) are required to maintain a margin with the clearing house. This margin is the amount of funds that clearing members must deposit and maintain to open a futures position, and is recalculated daily. After a margin call is made, clearing members typically have several hours to make a payment to the clearing house and fund the margin. If the margin is not funded, the position will be force liquidated, and would likely result in financial losses and impact to the clearing member's customers.

Clearing members will also typically perform their own margin calculation, and reconcile against the calculation from the clearing house. There are scenarios where the calculations differ, which could be due to different interpretation and codification of the logic from the rulebook, or where there are discrepancies in the data used.

02 Opportunity

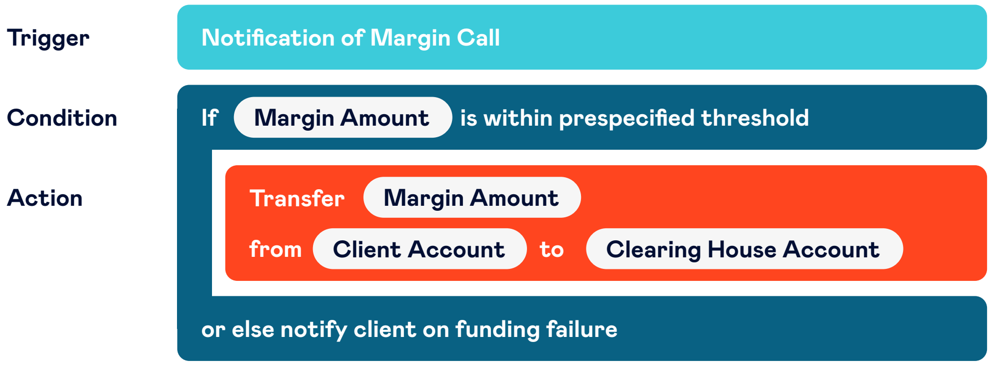
Automation of the margin funding process can reduce the processing time, and provide certainty for transactions. As the funding is processed in the bank's environment, there is reduced dependency on a client's systems and this further improves the certainty of payment. There is also a possibility of having a common data and calculation logic between the clearing house and clearing members, eliminating discrepancies in margin calculation and the need for margin reconciliation.

By reducing the processing time, the turnaround time from issuing a margin call to settlement can be significantly reduced. This creates a possibility of increasing the frequency of margin calls. Margin is used as a means of managing risks by the clearing house, with part of the risk due to the price volatility. Shortening the duration between margin calls reduces the volatility in between the calls. (The possibility of price changes exceeding a certain amount is much higher for a time period of a year vs. a day vs. an hour). Lowering the risks could also result in lowering of the margin requirements, reducing the amount of collateral that clearing members have to place with the clearing house, allowing for more efficient usage of the freed funds

03 Solution

The client, a clearing member, can set up a programmable instruction to make payment to the clearing house when a margin call is received. It is likely that the client will put in place certain safeguards, such as a variation threshold over the previous day's margin. If the amount is within the threshold, payment is made automatically to the clearing house. Otherwise, a notification will be made to the client to validate and manually release the payment.

Automated Margin Funding


Trigger
Notification of Margin Call
Condition
If Margin Amount is within prespecified threshold
Action
Transfer Margin Amount
from Client Account to Clearing House Account
or else notify client on funding failure

04 Possible enhancements

The margin calculation logic could be codified in the programmable instruction, with the payment amount automatically calculated. This would eliminate the need for reconciliation, as the same logic, codified in the same way, is used by both the clearing house and the clearing members. This is however, dependent on the complexity of the algorithms used, and the computational costs of doing so on the blockchain.

05 Benefits

This will reduce operational costs and reduce the need for reconciliation by the clearing members. In addition, increasing the frequency of margin calls could reduce the risks on the clearing house, and reduce the margin requirements on clearing members. This could reduce the collateral required to fund the margin, resulting in more efficient use of funds by the clearing members. These benefits are possible through cross-organization automation, due to the execution on a common platform with a common set of data.

Chapter —— 07

Conclusion and Future Exploration

In this paper, we summarized both client-side and bank-side programmability in the form of smart contracts on blockchain networks, and offered several use cases where the use of bank-side programmability could enhance the efficiency and reliability of the system. We presented several real-world use cases of programmability and highlighted the tradeoffs that programmable smart contract platforms provide in these cases.

We find bringing programmable logic into the bank-side transactor processor provides several key benefits including increased composability, efficiency and reliability. We present these benefits in each use case we cover.

We also saw several areas that call for future exploration. As the dividing line between clients and the bank continues to shift, technologies capable of supporting the safe externalization of nodes rises in importance. Contract expressivity has been initially limited for risk mitigation reasons, so more nuanced handling of these risks should open up new expressive capabilities for clients. Lastly, proof standardization could bring about additional features without having to significantly expand the trust domain of the system. Each of these categories are candidates for future exploration.

Node Externalization

The innovation of bank-side programmability lies in the ability for clients to deploy their programmable instructions with the bank, which is then executed in the bank's environment. This construct assumes a separation of the bank's and client's execution environment. J.P. Morgan is exploring the externalization of nodes, where clients could host JPM Coin nodes within their environment. In such a scenario, the separation is blurred, with clients being able to deploy their own applications in the form of smart contracts on the same environment as JPM Coin's smart contracts. The applications could interact directly, and in doing so, allow for better visibility and tighter integration of banking and payment processes with business processes. There are immense new possibilities, especially with the growing interest in unified ledgers that could enable interactions of multiple applications to support a variety of use cases.

Expressivity

Expressivity, or the number of features the programming environment supports, is deliberately restricted today as a governance mechanism to manage risk. This constraint can be relaxed with better understanding of the risk impacts of client code to be run in a bank's environment, and a more nuanced implementation of mitigation techniques.

For example, mechanisms for automated decompilation and review of programmable instructions could be added. Smart contract audits and automated ways to test interactions between programmable instructions, such as loop or infinite recursion detection using counters would also help address the risk concern. One could also limit the number of calls to particular contracts or sets of contracts.

Risk can also be mitigated through the use of libraries or Low Code or No Code (LC/NC) environments, which can improve expressivity while still imposing constraints to reduce syntax errors and bugs. Constraints can also be applied on a more granular level, through selectively enabling functions, operators and controls. There are different models for integrating LC/NC into code execution. One option would be to have LC/NC environments generate Solidity code, which would present opportunities to deploy the same logic across more than one blockchain. An ecosystem of third party LC/NC providers could supply this as well as pre-paid gas as a service.

Each of these categories would benefit from deeper investigation. This would create a more flexible, featureful, and trusted digital payments system

Portable Proofs

Programmability is greatly enhanced by trusted and validated data. Trust is conventionally conferred by establishing authenticated channels between parties through which data flows. An alternate strategy is to shift the trust in the channel to the data, and allow data to be shared in a secure and trusted manner, even when it is passed through unsecured channels. This could present a more flexible and scalable model of connectivity and information transfer, as transacting parties could be conveying information instead of needing the platform to build secure channels to all possible data sources.

For example, an IoT device might commit to the state of a shipment in a trade scenario by providing a signed message with the geolocation of the shipment and confirming that it has reached its destination. Or, a market data provider could provide updated prices or rates, such as the interest rate to trigger payment on an interest rate swap. Such messages may flow through untrusted channels on their way to the transaction processor. If the message is altered at any point in the journey, the signature will fail verification and the altered message would be dropped. Standardization of both messages and pathways for message delivery would lead to better efficiency, composability and reuse. Additionally, a single proof of execution from a remote system could be used many times without continually reattested accuracy, as may be required in conventional systems.

References: **A:** Parsec https://dci.mit.edu/parsec / **B:** Ethereum https://ethereum.org/

Disclaimer