Hyperledger Proposal for Transaction Execution Platform

Last edited: April 17th, 2019 (Original Google Doc: https://docs.google.com/document/d/13d0cMReGOhK13BbdgMOFZy_prUzqWBXWc4nlI7mehpY/edit)

Approved and tracking setup at Transact Execution Platform

HIP Identifier: Hyperledger Transact

Sponsors:

Shawn Amundson, Bitwise IO
Dave Cecchi, Cargill
Andrea Gunderson, Bitwise IO
James Mitchell, Bitwise IO
Peter Schwarz, Bitwise IO
Dan Middleton, Intel
Gari Singh, IBM
Jonathan Levi, HACERA

Abstract:

Transact is a transaction execution platform designed to be used as a library or component when implementing distributed ledgers, including blockchains. Hyperledger framework-level projects and custom distributed ledgers can make use of Transact's advanced transaction execution and state management to simplify the transaction execution code required within their projects or to gain additional features. Transact provides an extensible approach to implementing new smart contract languages called smart contract engines. A smart contract engine is generally an implementation of a virtual machine or interpreter which processes smart contracts. Examples of smart contract engines include Seth (EVM smart contracts) and Sabre (WebAssembly smart contracts). SDKs are provided for implementing smart contracts and smart contract engines. Smart contract business logic can be written in a variety of supported programming languages.

Context:

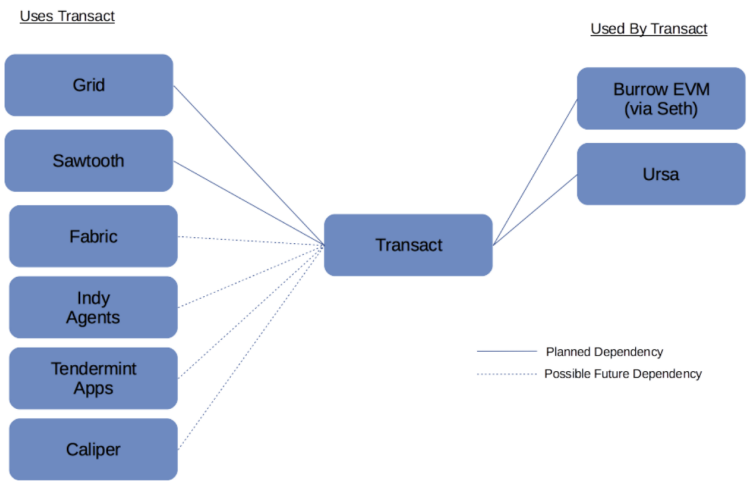
In distributed ledger frameworks, a transaction represents an intended change that is submitted by a user. Transactions are interpreted by smart contracts, which update the current state of the system as a result. Transact handles the overall mechanics of scheduling transactions, dispatching them to smart contracts (often referred to as "executing transactions"), and state management.

Transact has taken inspiration and architectural elements from Hyperledger Sawtooth's current transaction execution platform, including the approach to scheduling, transaction isolation, and state management. Transact extends these concepts with features such as a flexible models for execution adapters and database backends.

Hyperledger Grid will use Transact to execute Sabre contracts. Thus, frameworks that use Transact (or support it as an option), will be well-positioned to integrate with Hyperledger Grid.

Transact may also be used by other existing Hyperledger projects. For example, Transact's transaction receipts are similar to Ethereum's transaction receipts in purpose and will contain information necessary to derive Hyperledger Fabric read-write sets. As a result, it should be possible to add support for Transact alongside the current support for Fabric chaincode. The FAQ section of this proposal includes additional examples of possible relationships between Transact and other Hyperledger projects.

The following hypothetical dependency graph shows how Transact fits within the family of Hyperledger projects:



Like Hyperledger Ursa, Transact is a library that may be used ***optionally*** by projects, and no mandate is intended or desirable.

Motivation:

Existing solutions for smart contract execution are generally tied to a specific distributed ledger implementation and design, limiting their reusability. The sponsors desire a design that is substantially similar to the transaction execution platform within Sawtooth, but which is reusable as a library or component for building distributed ledger solutions. We believe that Transact will lower the threshold for developing distributed ledger solutions by removing the need to implement transaction execution code.

Dependent Projects:

This project does not depend upon a Hyperledger framework project.

Proposed Status:

Incubation.

Solution:

Transact is fundamentally a transaction processing system for modification of state. State is generally a Merkle-Radix tree, key-value database, or SQL database. Given an initial state and a transaction, Transact can execute the transaction to produce a new state. These state transitions are considered pure, because only the initial state and transaction are used as input. In contrast, other systems such as Ethereum combine state and block information to produce the new state. As a result Transact is agnostic about any particular framework feature, including the concept of blocks or a particular chaining approach.

Transact does not include other features which are commonly found in frameworks like Sawtooth and Fabric, such as the concepts of consensus, blocks, chaining, and peering. These features are the responsibility of the Hyperledger frameworks and other distributed ledger implementations, even if they use Transact for smart contract execution.

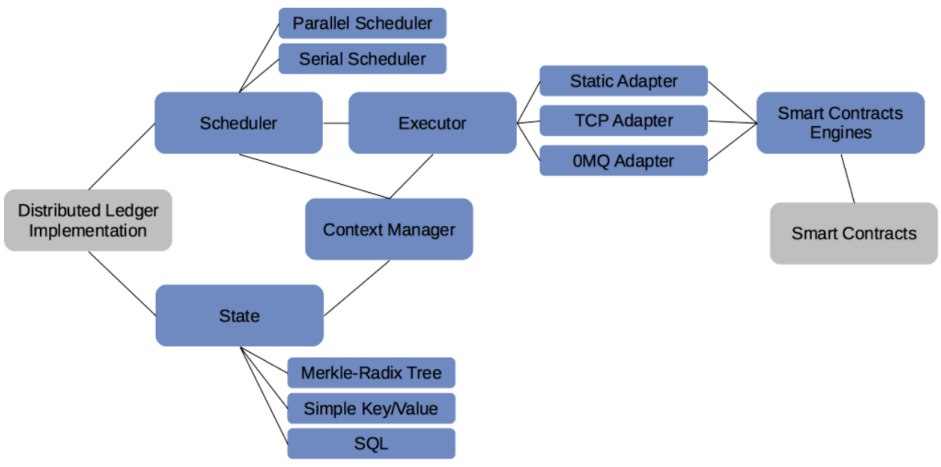
The following features are currently supported by Transact or are planned in the short-term (not a comprehensive list):

- Transaction execution adapters which allow different mechanisms of execution. For example, Transact initially provides adapters that support both in-process and external transaction execution. The in-process adapter allows creation of a single (custom) process which can execute specific types of transactions. The external adapter allows execution to occur in a separate process.
- Serial and parallel transaction scheduling. Serial scheduling executes a single transaction at a time, in order. Parallel scheduling executes multiple transactions at a time, potentially out of order, with specific constraints which guarantee a resulting state that matches the in-order execution that would occur with serial scheduling. Parallel scheduling provides a substantial performance benefit.
- Pluggable state backends. Transact initially provides a performant LMDB-backed Merkle-Radix tree implementation and an in-memory Merkle-Radix tree (primarily useful for testing). Key-value databases and SQL databases will also be supported in the future.
- Transaction receipts, which contain the resulting state changes and other information from transaction execution.
- Events that can be generated by smart contracts. These events are captured and stored in the Transaction receipt.
- SDKs for the following languages: Rust, Python, Javascript, Go, Java (including Android), Swift (iOS), C++, and .NET.
- Support for Sabre (WebAssembly smart contracts) and Seth (EVM smart contracts).

Transact architecture includes the following components:

- State. The Transact state implementation provides get, set, and delete operations against a database. For the Merkle-Radix tree state implementation, the tree structure is implemented on top of LMDB or an in-memory database.
- Context manager. In Transact, state reads and writes are scoped (sandboxed) to a specific "context" which contains a reference to a state id (such as a Merkle-Radix state root hash) and one or more previous contexts. The context manager is responsible for implementing the context lifecycle and services calls to read, write, and delete data from state.
- Scheduler. The Transact scheduler is the component which orders transactions to be executed. Concrete implementations include a serial scheduler and a parallel scheduler.
- Executor. The Transact executor obtains transactions from the scheduler and executes them against a specific context. Execution is handled by dispatching the transaction to specific execution adapters (such as ZMQ or a static in-process adapter) which, in turn, dispatch the transaction to a specific smart contract.
- Smart Contract Engines. The virtual machine implementations and interpreters that run the smart contracts.

The following diagram depicts the general relationship between the components within Transact:



Effort and resources:

The sponsor companies plan to devote engineering resources to the project, including development and documentation.

An initial stable release is anticipated in 2019. Because this project will provide the basis for smart contract processing in Sawtooth 2.0, Grid, and other projects, we expect this project to quickly mature.

How-to:

The project will be managed in a GitHub repository and will follow community norms present in other Hyperledger projects.

Clojure:

This project may support a Clojure SDK in the future.

Frequently Asked Questions:

1. Why are libraries a good idea for Hyperledger? There has been a lot of talk within Hyperledger about code sharing and re-use between projects such as Sawtooth, Fabric, and Indy. Libraries are a concrete way to work toward that reality.
2. Ugh, some people love this project's name, but I hate it. Will you *please* change it? Naming is difficult and it is certainly hard to come up with a name everyone likes. The sponsors intend to work with the Hyperledger marketing committee and others to select a name prior to the project's announcement. If you have a suggestion, please send it to one of the sponsors.
3. How is this project related to Hyperledger Sawtooth? A future version of Hyperledger Sawtooth will use Transact as a library to replace the existing Python transaction execution platform. To enable this adoption within Hyperledger Sawtooth, Transact will provide backward compatibility for existing Sawtooth transaction formats and Merkle-Tree state handling. However, the native transaction formats in Transact will differ slightly in order to meet Transact's cross-project requirements.
4. How is this project related to Hyperledger Grid? Hyperledger Grid uses Sabre (WebAssembly smart contracts). When Grid executes Sabre contracts, it will use Transact, either through Hyperledger frameworks or directly. Hyperledger Grid may also use Transact to execute Seth smart contracts in the future. By focusing on running smart contracts with Sabre and Transact, the Hyperledger Grid project intends to make it easier to re-use Grid components.
5. How is this project related to Hyperledger Fabric? Transact could be integrated into Fabric as a sibling to chaincode. Fabric read-write sets can be derived from Transact's transaction receipts. In addition, Fabric's existing chaincode could be supported by Transact.
6. How is this project related to Hyperledger Ursa? This project will include pluggable support (not implementations) for signing and hashing, with built-in support for Ursa. As Ursa matures, we anticipate that it could become the default approach to signing and hashing within Transact.
7. How is this project related to Hyperledger Indy? Today, Indy does not have a general-purpose capability for processing smart contracts. As a future feature, Transact could be used to add transaction-based smart contract processing to stateful Indy agents.
8. How is this project related to Hyperledger Burrow? Seth (EVM smart contracts in Hyperledger Sawtooth) uses Hyperledger Burrow's EVM implementation to execute transactions. Transact will support Seth and thus Burrow EVM execution.
9. How is this project related to Tendermint? There is a strong overlap between what Tendermint considers an ABCI application and the features provided by Transact. Thus, Transact could be used to implement ABCI applications.
10. Should this project go to labs? Conceptually, this project has gone through several iterations within Sawtooth, where the ideas have been proven to work. The sponsors feel it is important that this component find a permanent home prior to adding it as a dependency in Sawtooth and Grid, in part because of the importance of this project within their architectures.
11. Why not maintain this project outside of Hyperledger? The sponsors of the project work on other Hyperledger projects and currently feel that Hyperledger is the best place for this project.
12. Why not put Transact in a Sawtooth repo? As we consider the dependency structure between projects, having this specific capability within Sawtooth would require the other projects to depend upon Sawtooth if they wish to adopt Transact functionality. While this may be acceptable in theory, it can be undesirable to use a component within a larger project because the component will primarily be driven by the larger project, not independently. The sponsors desire Transact to be independently consumable by other projects and to be a peer with Sawtooth when it comes to driving project direction. For example, Grid use-cases should drive future Transact features and releases on equal footing with Sawtooth.
13. Can this project be used for smart contracts which require block information? Yes. Although this project's APIs only support pure state-transition functions, fields in the execution protocols and APIs will allow frameworks to add additional framework-specific capabilities such as access to block information.
14. What programming language will be used for implementation? Most of the code in Transact will be written in Rust. SDKs will be provided for a variety of programming languages.
15. Is it an application? Definitely not. Totally not.
16. So... where is the code? The core Transact library is currently available at: https://github.com/bitwiseio/transact