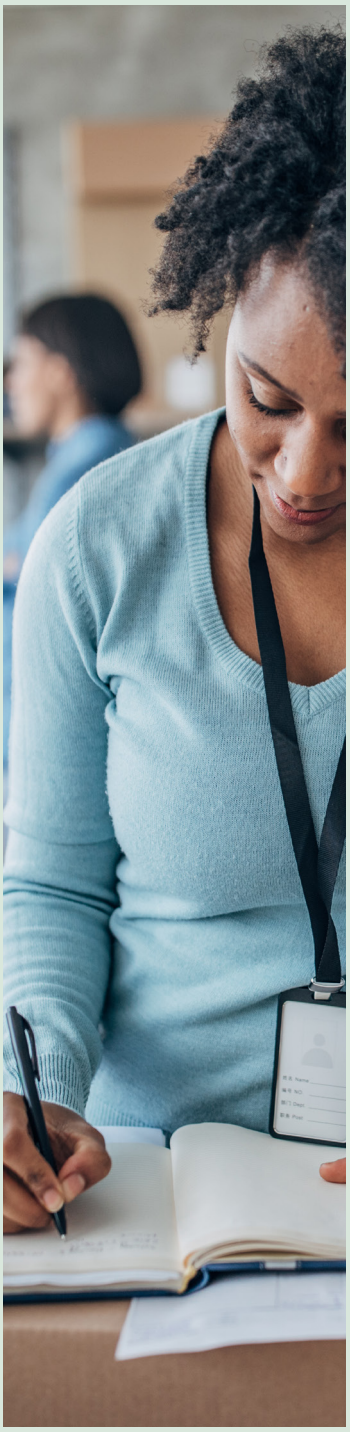
WHITE PAPER



LEVERAGING BLOCKCHAIN IN CUSTOMER ORDER MANAGEMENT (TELCOS)

Infosys®
Navigate your next

Summary

This paper is focused on lifecycle management challenges faced in Telco organizations for Orders that may span several physical and logical Services/Elements and require interactions from multiple stakeholders (internal and external). An entire spectrum of Systems and Processes (which lead to data duplication and $$$) are invested to salvage trivial ask for tracking (Orders->Services->Assets).

Using this proposed approach on extending standard Blockchain implementation (such as Ethereum, Hyperledger etc.) to this use case, **we can bring in the best of both worlds** (Centralized monolithic and Distributed system patterns) **to existing ecosystem.**

It also makes provisions to **bring maturity** in business process management with time, by utilizing an amendable and agreed on **Constitution within a Consortium.** Although it can be applied to any industry with Supply chain requirements, the use case we are taking here deals with, Orders for Telco Customers (Government, Enterprise, and Retail).

Please note that solution applies to consumer world too.

What is different here

In a typical Consumer world scenario, primary concern is "Cycle time". Say, for Mobile and Home broadband services, it is in days, may be a few weeks. However, with Customer Segment (Gov, Enterprise, Retail), Order cycle time (for O2A, Order to Activate) spans in months and in some cases, Years, just for completion of one (Master) Order.

In Customer Segment, **ONE Order** can be a very big number.

One such order can easily contain multiple sites, multiple services with internal/ external dependencies and hundreds of parts (both Managed and Unmanaged, plus Accessories). Normal add-ons will include Amortisation, Upgrade path support, Fault repairs (and the list grows). So, it is Large and Complex. What we observe here (because of sheer size & complexity) is, following inefficiencies arise that are systemic in nature and we need a good resolution approach –

1) It is always difficult to estimate stock consumption and then track it (What was reserved, and did it get used for that purpose, and how much?). Replenishment becomes a challenge, so should we overstock/understock? It may not be always directly a Telco problem. This will be more of a Vendor end challenge, but it hurts margins for both parties.
2) In this Segment, Sunny day scenarios are very less. Most of the times, we need to Change/Postpone/Cancel/Recover multiple items in very single life of One Order. This creates two types of impacts -
   a. Increase in Complexity - Process gets complex. Communication (unsupported interface scenarios) and Turnaround timelines suffer.
   b. No single Point of truth - What is reported in different systems, and to what degree do they differ from one another (which way to reconcile)?

This becomes standard (a.k.a. BAU, Business as Usual) where manual updates/overrides are used to fix these (comments are "conveniently" added to explain what is being fixed). Unfortunately, there is no way to tell how big of a leak is (and there will be multiple of these).

Conventional approach

A logical way will be to put all of it in one Master inventory and always keep it up to date (somehow). **It does work** for certain scenarios, where there is only one party/Reseller involved.

But in large Telco solutions there are multiple Vendors, Suppliers and multiple contract management systems which effectively create multiple "Masters" (pseudo/local masters). This is the base challenge of being a Telco. It must support Technologies that are 2 generations old and at the same time, it must invest in technologies of future (5-10 years down the line). So, there are Services with Customers that must be supported through older Inventory/ hardware channels, while using other systems, it will have to manage fulfilment of the latest and greatest on offer.

Invariably, a Telco enterprise ends up with all kinds of systems and processes that must be baked in together to track and deliver using different networks and technologies. This topic gets big and messy very quickly (and we completely skipped diving into complexities of distributed Service and Resource inventory management).

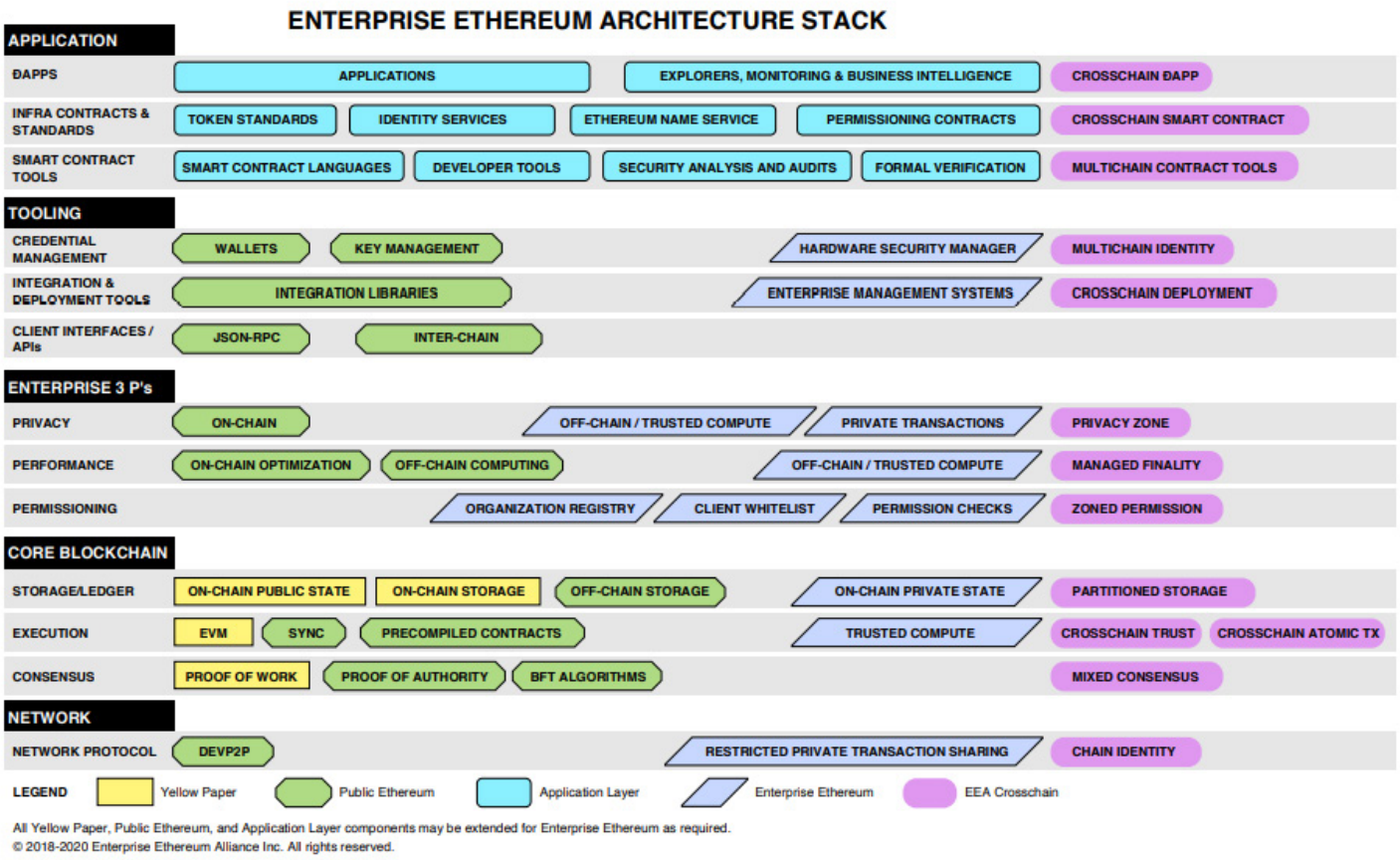
So, Single Master Inventory (Point-of-truth) option is not practical. Let's see if using Blockchain brings anything new to the way we tackle these problems.

Blockchain adoption in enterprise

Let's first quickly summarise the current state of adoption in Enterprise and the "quirks" associated with this approach. Though, Blockchain is now well accepted in other industries (Supply Chain, trace, finance etc.), it is still bit unconventional when we talk about OSS (Telco world). Hence, we have this write-up.

In Private or Consortium based deployments, there are fundamental differences in terms of trust of participants. Parties are well known, usually legally registered institutions. There are natural (financial) disincentives for bad behaviour. There are established legal remedies - whether imposed by a Central authority (Government) or agreed to by a binding off-chain agreement.

Enterprises are also providing feedback that they expect to be using more than one blockchain (Crosschain implementations) in their business, often different blockchain types.



Source reference URL - Crosschainsecurityguidelines (entethalliance.org)

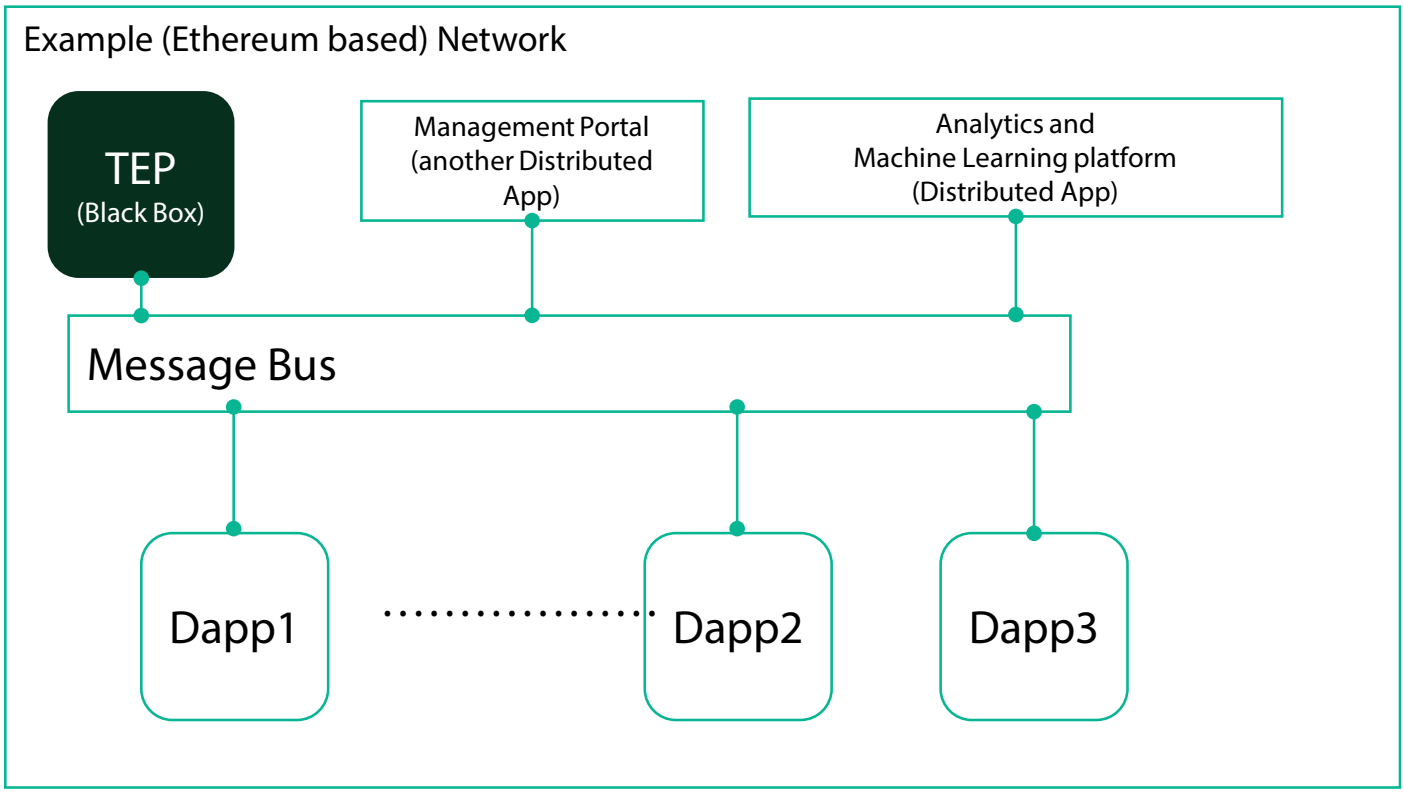
To mitigate Privacy related issues, a whole segment of solution patterns are available. Adding references to few of them below -

- Privacy-Preserving Scheme in the Blockchain Based on Group Signature with Multiple Managers (hindawi.com)
- Ethereum vs Fabric vs Corda: Enterprise Blockchain Protocols Compared (kaleido.io)
- Zero Knowledge Proofs: An illustrated primer – A Few Thoughts on Cryptographic Engineering (cryptographyengineering.com)

An Enterprise also expect to interact with other enterprises having their own blockchain types. **In addition to Separation of concern, there is a need for sharing of the logic code across blockchain types.** This may require sharing of same logic in different, distinct chain deployments OR to have this logic bridge multiple chains together. A model where the same logic must be written, tested, and maintained in different chain specific languages is not optimal.

Blockchain domain is vast and offers many ways to tackle these scenarios. Here we will be using Enterprise Smart Contracts and Cryplets for the required flexibility in building a Trusted, yet Discreet Transaction model.

A trusted, yet discreet transaction model



What we see above is a simplified view which most of us can co-relate to a standard Pub-Sub (Event driven Architecture) Order orchestration model in OSS (Telco) space. Familiarity is intentional here. This model inherently allows us to be readily accepted/adopted in Enterprise and grow organically based on changes in ecosystem.

Only differentiation here is that **Black box** which is responsible for all the magic we are planning to bring in. Well, that and how all of this gets baked in together.

Characteristics of this transaction model

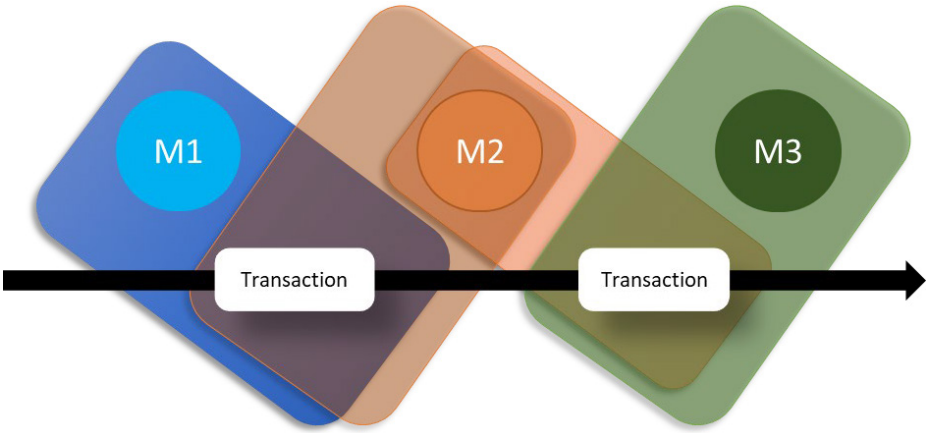
We will briefly touch upon **key characteristics** that enable us to deliver the desired features -

- Consortium(s): Supports 1 or more, depending on how many tenants (business streams) would like to use the same infrastructure
- Constitution: A Constitution for every consortium. This defines policies on What and how (i.e., Roles, Permissions, Allowed Actions with scenarios, including Voting rules/types). Policies can be queried and modified (and are treated as Actions). Other examples of Action include: Authorisation and posting encrypted notifications to allowed members in the Consortium.
- Smart Contracts: Functions which execute the Constitution actions, like read, add members, update etc.
- A Trusted Execution Platform: Where Smart Contracts are executed. This could be an appliance hosted by a Central agreed/trusted authority. This platform can also be implemented using specialised software stack. There is no persistence of transactions and therefore, we might use Analytics platform as an "approved" sink to all the applicable events.
- **Critical requirement:** This Trusted platform (Black box) complies to always encrypted Data i.e., "at rest, in motion and in use". Example: It cannot be implemented using a simple VM, since Hypervisor will have access to data in memory i.e., "in use". Messages can only be decrypted by this trusted platform. Hence, the secrecy/segregation of information is implemented by the Platform and governed by its constitution.
- Analytics and Machine learning loopback provides insights (as per Constitution) on how the transactions are being utilised, its drill down details (where applicable) and ability to identify any attempts of possible misuse to the platform.
- There is no compute cycle wastage (say Currency mining) unless unauthorised access to Messaging service has been obtained. This pattern should be detected early enough by the Analytics platform.

Key feature - support for visibility constraints

Simple flow where multiple Members of consortium transact on same object (M1 ----> M2 ----> M3)

- M1 may allow M2 to see the transactions made by M1 (but M3 can't access it)
- M2 can make a new transaction on **same object** and allows M3 to see this new transaction, (but M1 cannot).
- When M3 makes an Update, it may allow M1 and M2 both to see the changes it has made. However, previous two transactions remain hidden as they were.
- So, how do we trust any transaction (if the history of its changes cannot be seen, due to constraints in visibility)? A transaction is always validated by the Trusted platform and hence, members can operate in discretion.



Note: This allows for all types of possibilities in Telco where multiple Vendors can participate in fulfilling the same order without sharing any information with competition. On top of that, information is available on the Assets and Order at relevant milestones only.

A constitution allows for growth and management

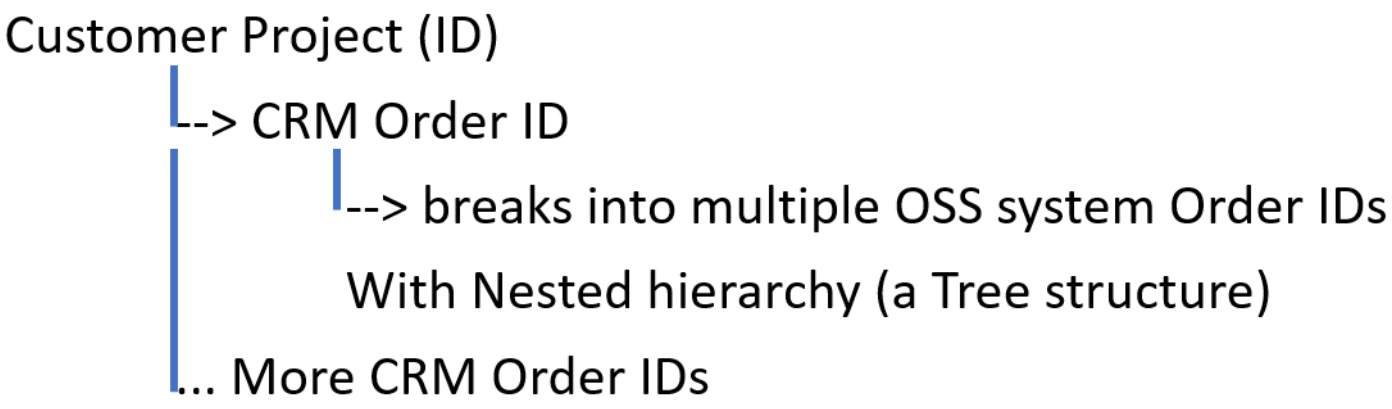
Following basic constructs are used to make system flexible to changes -

- A constitution holds the policy for each Action type and new actions can be added/ modified/ deleted, as needed.
- Voting may be associated for some Action types, like adding members, or changing a member's permission. A transaction/action type can only be accessed by the applicable consortium members where permissions are granted. All these functions are always executed on the "Black Box" (a.k.a. TEP, Trusted Execution Platform).
- Members will vote (as per Constitution rights) to make amends to the Actions and behaviour. Effectively, additional code (smart Contracts) can be imported at runtime.
- Secure protocol must be used for all messages and communication.
- Expected typical transaction rates for such a system is expected to be north of 1,000-1,500 transactions per second. This should easily satisfy business needs for an Enterprise.
- Trusted platform can be managed internally as cluster of resources for mitigating any security concerns. However, this is not always encouraged as the design complications encompassing Security and Consistency must be adequately addressed.

Note: More insights on this gets covered in Appendix – Internals of Transaction flow

Applying this transaction model to customer order management

First, typical anatomy of a customer order



Note: For efficiency, at a later stage some of the Orders may be executed in batch where a logical grouping might entail same location/ technology within Same Project Identifier.

There could be several seemingly disjoint work parcels flowing through various systems but can be still aggregated based on one common identifier. This common "Root Identifier" can thus be trusted to use for dependency/co-relation. There are practical challenges to this ->

1. While this "Root Identifier" binds everything together, the immediate children (CRM Orders) will have their own independent lifecycle. Ideally (and it is a presumption), Start date and End date of a Project will encapsulate the life cycle of all other (children) CRM Orders. So, can this affect the "Asset Lifecyle Management"? If Asset needs to be replaced, for whatever reason, will it be associated to same Project ID? Necessarily not. This depends on lots of factors. So, tracking and co-relation for follow-up orders OR Revisions is not black and white in nature.
2. Continuing previous point, status of Asset is loosely related to status of Service(s). If the Service(s) is/ are not activated and tested by the field agent, then the Asset can only achieve status of "Delivered/received" at Customer Premises. An actual Service activation test will determine whether the Device is good to be called "commissioned/in-use".
3. We must determine the most appropriate stage to accurately resolve/identify the existence of an Asset. And how important it is, to tightly couple an Asset Identifier with Device Physical Identifier such as a Serial number. The deployed Asset is a logical view and is subject to ad hoc Service repair requests that might not flow through all key systems. So, what kind of Lifecycle Management are we interested in? Is it for the Asset OR is it for the Physical Device?
    a. Customer is keen on Asset (doesn't care about Serial# of the device).
    b. Device Vendor doesn't care about Asset. Serial# is key to track what is delivered.
    c. Telco Service Provider cares about Asset and to an extent, about the Serial# because of the warranty procedures (mostly when they might deal with Vendor for replacement).

So, we add LCM model assumptions for asset

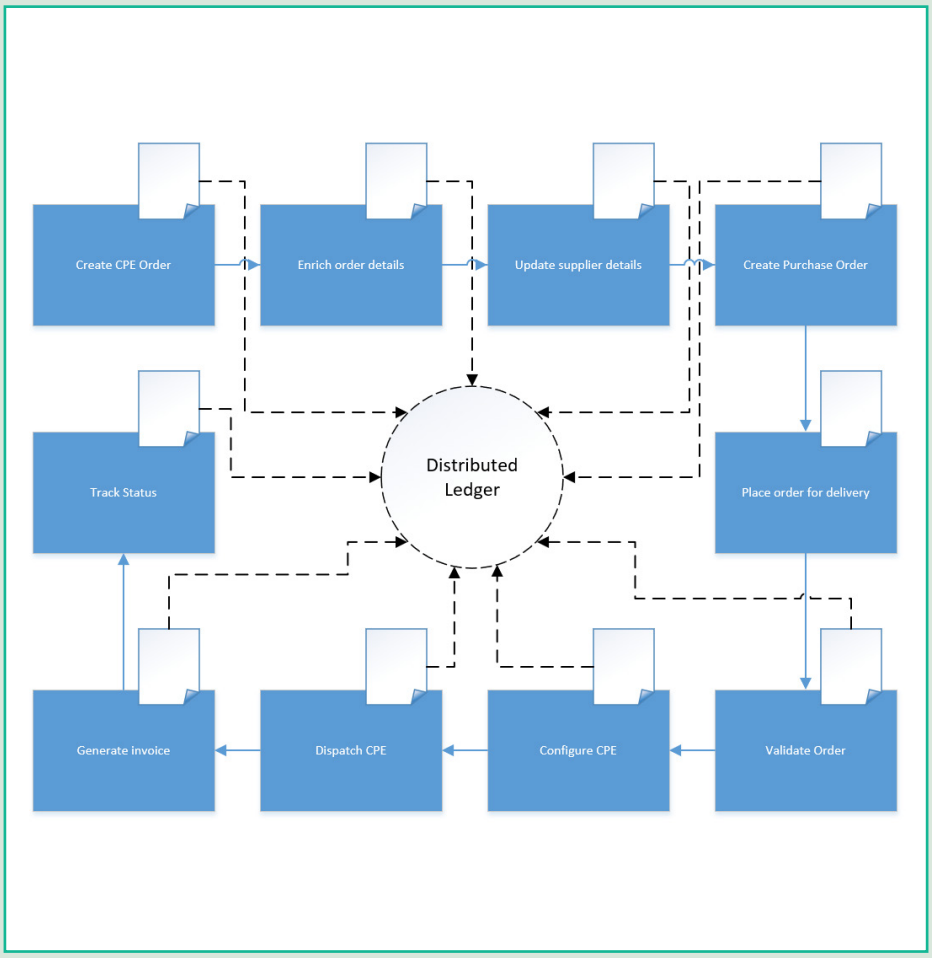
1. An Asset in its lifecycle can and will be associated with multiple "Root identifiers" and Children work parcels.
2. Changes in design due to requirement correction can also directly impact the Asset lifecycle status.
3. Most important thing is association of Asset->Serial#. For maximum flexibility and minimum complexity, we will some base assumptions -
    a. An Asset (ID) can be created in concept, with/without a Serial# attached to it.
    b. However, the lifecycle status of Asset (ID) cannot progress beyond a certain point without associating with a Serial# of the Device.
    c. Once a Serial# is associated to an Asset (ID), the Asset must carry this association till the Asset is decommissioned.
    d. There should be a trusted way to determine a valid/unique association between the two (Asset ID and Serial#). So, this **association must be executed by agreed member(s) of the consortium.**

Test approach resiliency

Managing Action Rollback (i.e., Change and/ or Rejection of Change).

Say, a Single Order carries 5 logical Services (A, B, C, D and E).

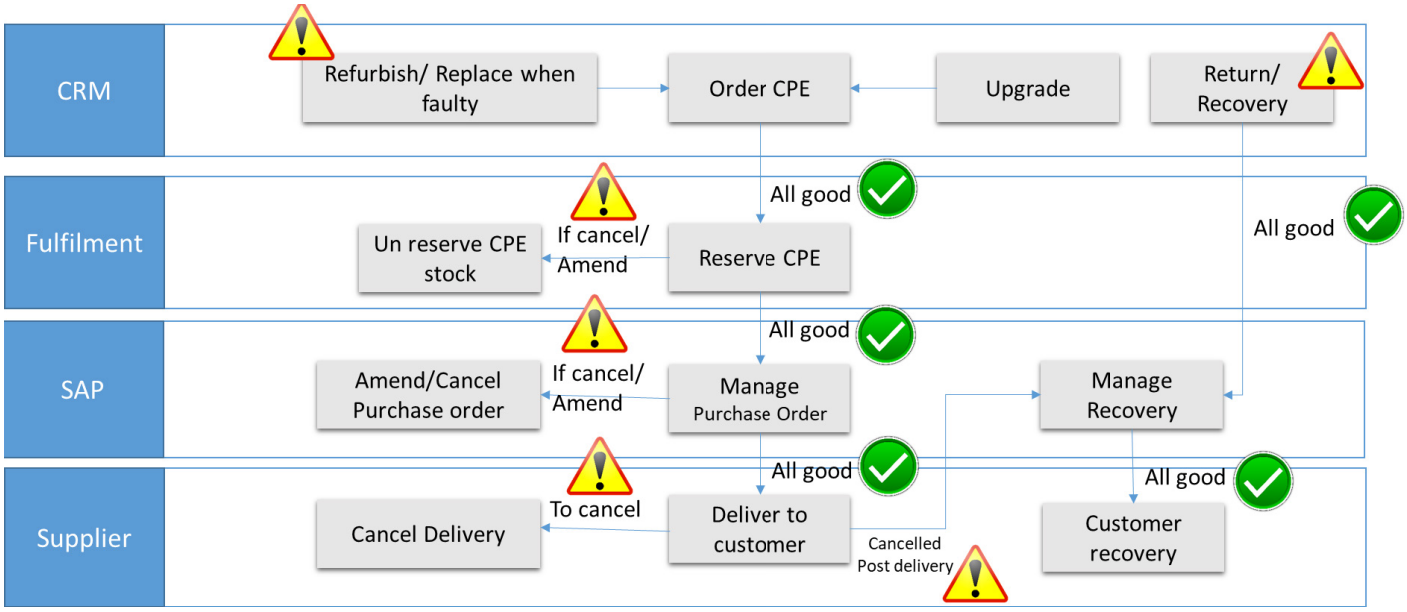
- As part of design (late in Order lifecycle), multiple Devices were added as requirement. They are A1, D1 and D2
- Now, we applied Order requirement changes (due to any reason, previous design cannot be implemented OR simply change in requirement). This creates different Service versions, A', B, C', D (and no E).
- To make things messy, the eventual design takes out A1 and D2 devices, but adds C1 to the tally. So (A1, D1 and D2) becomes (C1 and D1).
- There are two options that can be exercised before all these changes are applied -
  - **Pause:** We can publish immediately to the intended Suppliers that a change might be initiated in CRM, so, this Order is being "re-planned".
  - **Auto-correct:** We may have enough visibility of potential impact to decide that we don't need to pause, and we will auto-correct, once design is redone (i.e., only additional devices may need to be ordered)



Note: Both options can be applied atomically on per Device basis in the same order. In this case, requested (set) metadata will drive decision making for Smart contracts.

Under all its discretion and trust, "new" OSS arrangement still appears simplified to its consumers -

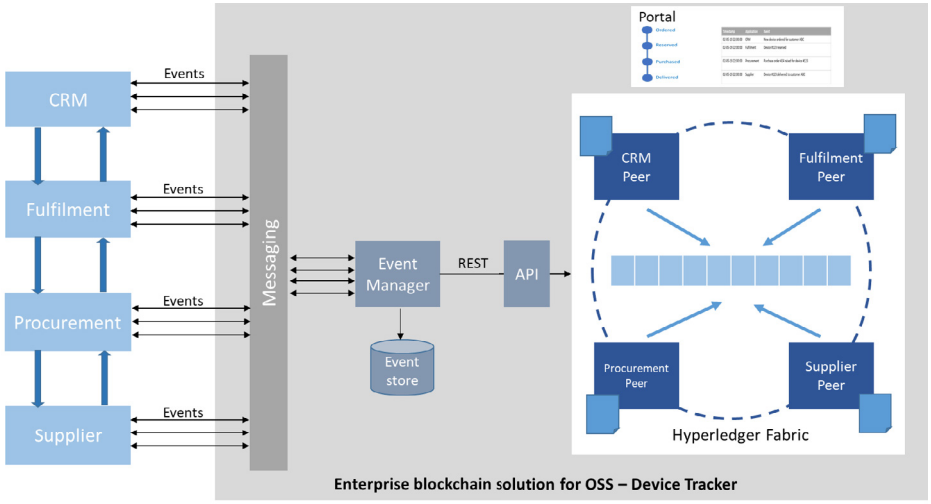
Sample flow



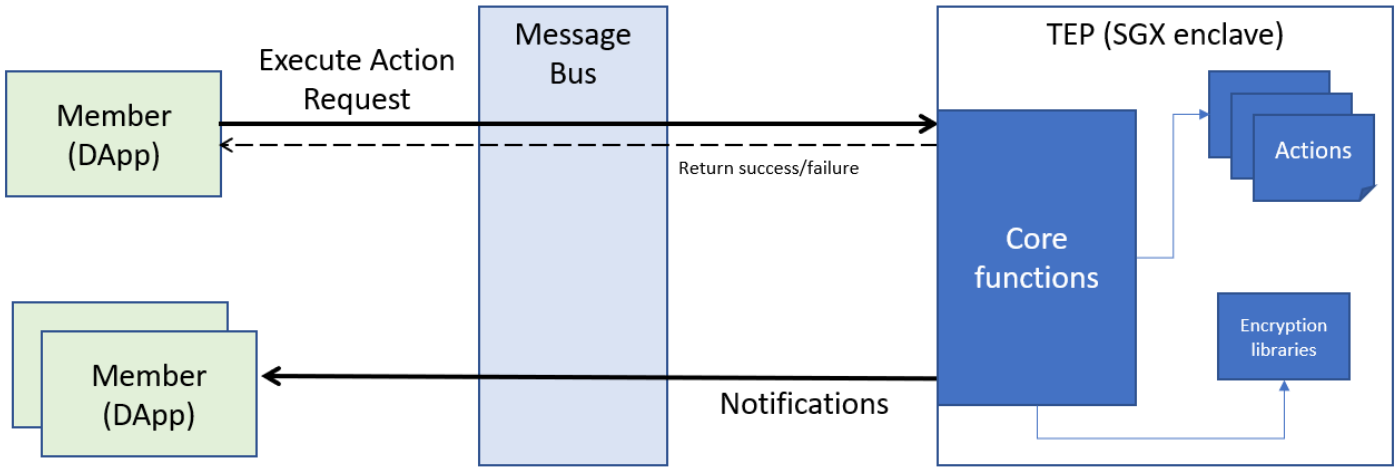
Note: Not all systems are required to be engaged in every flow. There is no PoNR (Point of No Return) required. Even if cancellation happens post device delivery, a separate process can be triggered for recovery. Business process will touch minimum relevant systems.

However, the status will be visible to all systems **(allowed interested parties)** to trigger any further processes (manual or automated).

It makes intra system status update interfaces (like status update notifications) redundant.



Appendix - internals of transaction flow



Flow example -

1) Member performs "executeAction" with Action name "updateDeviceOrder" and provides supporting parameters.
2) TEP receives the requests

a) Validates Permission to execute, returns success/failure for execute permission

b) Executes referenced algorithm which then may change the values and use sendNotifications to inform members as needed

3) Applicable member(s) receive Notifications on action and will follow local procedures as needed

To implement this, members need to support following methods (Common implementation library can be circulated/ shared amongst consortium members)

- async getNotification(action, parameters) - This is populated with values when TEP (trusted platform) executes "internal" Action sendNotifications and is sent to applicable members only. Once a notification is received, it is responsibility of the member to call appropriate executeAction method, as needed.
- async executeAction(action, parameters): For simplicity, execution of only one operation type is supported for each member. It is always only directed to TEP (other members have no use for this action). Only returns a success/failure message with Transaction ID (optional). Any output is received via getNotification (triggered by TEP). More details: Execute Action is used to Update Constitution and Assets.

Implementation details of executeAction on TEP end. Parameter info –

action: Action name (example: addMember, amendConstitution, refreshTrustKeys, updatePermissions, voteFor, getDeviceStatus, updateDeviceOrder etc.). This will execute existing Action in the constitution. TEP will verify permission for execution. Business validation is part of Action algorithm. An Action algorithm may then refer to additional Actions (Scope: Internal only. Please see amendConstitution Action parameters below). It should be noted that certain Actions are created as base/ fundamental Actions to support day to day operations. While they themselves can be modified/refined, it is advised to be done with maximum caution to maintain constitution integrity. Example of such actions -

amendConstitution: Adds/Replaces with new Action definition in the constitution. Always replaces the existing Action definition. Parameters include Language (example python/ java with version), Permission meta-data for members and actual Code submission. TEP will verify integrity of Action scope. We can use this to revise action definitions and/or Permission meta-data for members. amendConstitution Action must carry three parameters-

1. Scope (of type Public/Internal) -

   Public: Can be called by any member (subject to Permission meta-data validation by TEP)

   Internal: This can only by executed by TEP and is meant to be used as support algorithm for "public" Actions like gathering data. Outgoing calls might be made, if needed (example, sendNotifications or query external systems). An internal action must not reference another "Public" action. This is to restrict scope of chaining of actions.

2. Action Permission list - It is a simple list of: MemberID, Execute(Y/N) and Validity date range (Optional). Sub level checks like whether a member can change device status from A to B should be implemented within Action algorithm

3. Voting schema - Includes Voter list (specific/all/any), Majority margin (minimum count) and Veto member list. Voting will be conducted when the applicable Action is requested for execution (example - addMember). If voting is not needed for that action, we can specify Voter list as "Any", Majority margin as 1 and Veto member list as NULL. In such case, member requesting executeAction (subject to Permission list) will be counted as "Any" with default Majority margin count of 1. If "specific members" or "All" is used, then executeAction for that Action will not proceed till it is Approved or Rejected.

Note: Although amendConstitution can be used to disable/decommission any action, it is advisable to use updatePermissions to disable further execution permissions. As a special case, amendConstitution may replace itself (should the capability is provided, but it is highly recommended to avoid using such capability). In Enterprise scenario, normally one member will have permission to execute this Action. Same with updatePermissions (see below)

updatePermissions: Simple operation to updatePermissions for any action. It should not have capability to change permissions on itself. amendConstitution should be used to make such changes.

refreshTrustKeys: Provides ability for members to refresh trusted keys with TEP for communication. Initial set of trust keys are already added by addMember action (by a registered member) and therefore, any new member should use this method to refresh with new set of keys, as needed.

voteFor: This is a very special Action and is called by members in response to "getNotification" (where voting action is mentioned). It has dependency on internal Action "approveAction" to be implemented first. approveAction will verify the voting count and return the control back to original executeAction once a decision is reached.

parameters: This provides a flexible and powerful extension to implement any generic business validation algorithm. It is like declaring parameters for any method and should suffice to provide for any/all required functionalities under the specified constraints. A custom Transaction reference ID can be supplied to co-relate with getNotification responses.

Additional references

- GoQuorum (consensys.net)
- Enterprise middleware for blockchain smart contracts - Second State
- Growing interest in Telecom Sector - Blockchain In Telecom Market By Offering, Organization Size, Provider Type, Application, Region | Absolute Markets Insights
- Using Trusted Compute - https://entethalliance.org/wp-content/uploads/2019/11/EEA_Off-Chain_Trusted_Compute_Specification_v1.1.pdf
- Privacy-Preserving Scheme in the Blockchain Based on Group Signature with Multiple Managers (hindawi.com)
- Ethereum vs Fabric vs Corda: Enterprise Blockchain Protocols Compared (kaleido.io)
- Zero Knowledge Proofs: An illustrated primer – A Few Thoughts on Cryptographic Engineering (cryptographyengineering.com)

About the Author

Debashish Mukherjee
Senior Technology Architect

For more information, contact askus@infosys.com

Navigate your next