ETSI GR PDL 004 - v0.0.7 (2020-06)

**Group Report**

Smart Contracts in Telco Permissioned Distributed Ledgers — System Architecture and Functional Specification

Release

***Disclaimer***

The present document has been produced and approved by the <long ISGname> (<short ISGname>) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.  
It does not necessarily represent the views of the entire ETSI membership.

Reference

DGR/PDL-004

Keywords

blockchain, Smart Contracts, Policy, Service Level Agreement (SLA)

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C

Association à but non lucratif enregistrée à la

Sous-préfecture de Grasse (06) N° 7803/88

***Important notice***

The present document can be downloaded from:  
[http://www.etsi.org/standards-search](http://www.etsi.org/standards-search#Pre-defined Collections)

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:  
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.

The content of the PDF version shall not be modified without the written authorization of ETSI.

The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2020.

All rights reserved.

**DECT**TM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are trademarks of ETSI registered for the benefit of its Members.  
**3GPP**TM and **LTE**TM are trademarks of ETSI registered for the benefit of its Members and  
of the 3GPP Organizational Partners.  
**oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and  
of the oneM2M Partners  
**GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Contents

Intellectual Property Rights 5

Foreword 6

Modal verbs terminology 6

Executive summary 6

Introduction 6

1 Scope 6

2 References 7

2.1 Normative references 7

2.2 Informative reference 7

3 Definition of terms, symbols and abbreviations 7

3.1 Definitions 7

3.1.1 Smart contract 7

3.1.2 On-chain Smart contract 7

3.1.3 Off-chain Smart contract 7

3.1.4 Master-chain 7

3.1.5 Side-chain 7

3.1.6 Ricardian Contract 7

3.1.7 The Difference between Ricardian contract and Smart contract 8

3.1.8 51% Attack: 8

3.1.9 Eternal Contracts: 8

3.1.10 Coin 8

3.1.11 Token: 8

4 Abbreviations 8

5 Introduction to Smart Contracts 9

5.1 Introduction 9

5.2 Smart Contract Programming Paradigms 9

5.2.1 Object-Oriented Paradigm 9

5.2.2 Properties 10

5.2.2.1 Immutability 10

5.2.2.2 Availability 10

5.2.2.3 Transparency 10

5.2.2.4 Self-Execution 10

5.2.2.5 Reusability 11

5.3 Storage 11

5.4 Life cycle of a Smart Contract 11

6 Smart Contracts – Planning, coding and Testing 11

6.1 Introduction 11

6.1.1 Planning Phase 12

6.1.1.1 Ownership of a contract 12

6.1.1.2 On-chain deployment 12

6.1.1.3 Off-chain deployment 12

6.1.1.4 Immutable deployment 13

6.1.1.5 Terminatable deployment 13

6.1.1.6 Upgradeable deployment 13

6.1.1.7 Draft Template 13

6.1.1.8 Terms Negotiation: 14

6.1.1.9 Compile Draft: 15

6.1.1.10 Review: 15

6.1.2 Coding and Testing phase 15

6.1.2.1 Introduction: 15

6.1.2.2 Code/Programming language-level Testing 15

6.1.2.3 Smart Contract specific Testing 16

6.1.2.3.1 Open Source SC Analysers 16

6.1.2.3.2 Sandbox Testing 16

6.1.2.3.2.1 Three passes: 16

6.1.2.4 Validation 17

6.1.3 Deployment and Execution Phase 17

6.1.3.1 Deployment 17

6.1.3.2 Execution 17

6.1.3.3 Upgrading(discussed earlier – can be removed from here) 17

6.1.3.4 Termination 17

7 Architectural requirements for Smart Contracts 18

7.1 Introduction 18

7.2 Architectural requirements 18

7.2.1 Resuability 18

7.2.2 Self-Destruction 18

7.3 Reference Architecture 18

7.3.1 Introduction: 18

7.3.2 Reference Architecture 18

7.4 Transaction Dependencies 19

8 Smart Contracts – Applications, solutions and Needs 19

8.1 Introduction 19

8.2 Applications 19

8.3 Solutions 19

8.3.1 Introduction 19

8.3.2 Scalabliity 19

8.3.2.1 Check-point 19

8.3.2.2 Extensibility 19

8.3.3 Example: Smart contracts with QoS monitoring 20

8.4 Needs - Requirements to build a viable system with Smart contracts 20

8.4.1 Regulatory Aspects 20

8.4.2 Security of the Contracts 20

8.4.3 Enforceability 21

8.4.4 Availablity 21

8.4.5 Attack 21

8.4.6 Integrity 21

8.4.7 Risks 21

9 Limitations of Smart Contracts 22

9.1 Inter and Intra system threats 22

9.1.1 Introduction: 22

9.1.2 Absence of Termination clause/Self-Destruction 22

9.1.3 Accountability Control 22

9.1.4 Accountability Management of Smart Contract 22

9.1.5 Poor Exception Handling 22

9.2 Limitations 22

9.2.1 Introduction 22

9.2.2 Occupancy: 23

9.2.3 Latency 23

9.2.3.1.1 Underlying and Relying ledgers in permissioned context: 23

9.2.4 Not every term can be translated to a Smart Contract(Remove this one?) 23

9.2.5 Legal Uncretainity: 24

Annex A: Title of annex 25

Annex B: Title of annex 26

B.1 First clause of the annex 26

B.1.1 First subdivided clause of the annex 26

Annex C: Bibliography 27

Annex : Change History 28

Document History 29

# 

# Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server ([https://ipr.etsi.org](https://ipr.etsi.org/)).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or maybe, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

# Foreword

*[ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx)* [(](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx)*[EDRs)](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx),*

This Group Report (GR) has been produced by ETSI Industry Specification Group Permissioned Distributed Ledger (PDL).

The present document is part <i> of a multi-part deliverable. Full details of the entire series can be found in part [x] [Bookmark reference].

# Modal verbs terminology

*[ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx)* [(](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx)*[EDRs)](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx),*

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx) (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in the direct citation.

# Executive summary

This present document specifies a high-level functional abstraction of PDL Smart Contract System Architecture. In particular, basic building blocks for designing, coding and testing Smart Contracts for the PDLs. This includes describing how different classes of systems interact with Smart Contracts. Processes, models, and detailed information are beyond the scope of the present document.

# Introduction

The present document defines a high-level functional abstraction of policies to design and code Smart Contract components. Smart Contracts are mere codes, and if not well planned, designed, coded and tested; can leave the system vulnerable to external attacks and internal errors.

# Scope

**Scope of work to be undertaken:** The present document specifies the functional components of Smart Contracts, their planning, coding and testing. This includes:

a) reference architecture of the technology enabling Smart Contracts – the planning, designing and programming frameworks

b) specify how to engage using this architecture – the methods and frameworks the Smart Contracts building blocks possibly communicate

c) point out possible threats and limitations

# References

## Normative references

## Informative reference

# Definition of terms, symbols and abbreviations

## Definitions

### Smart contract

As per ISO /TR23455/2019 definition, Smart Contract is:

***It is a computer program stored in a distributed ledger system, wherein the outcome of any execution of the program is recorded on the distributed ledger.***

***A smart contract might represent terms in a contract in law and create a legally enforceable obligation under the legislation of an applicable jurisdiction.***

In this document we separate smart contract and smart legal contract.

### On-chain Smart contract

The on-chain smart contract is the contract that resides in the master-chain and is executed directly without the instantiation of any other contract. The beneficiaries get rewarded immediately, as soon as the contract is executed without the involvement of any other contract.

### Off-chain Smart contract

Off-chain contracts are the smart contracts stored away from the ledger(i.e. trusted database or side-chain) and their execution may depend on on-chain contracts(i.e. on-chain contract can initiate off-chain contracts) and later the state can be updated.

### Master-chain

The primary chain where the executions of the smart contract are recorded.

### Side-chain

The chain(s) which work as a secondary chain to main chain/ledger. It can be sused to off-load some of the computations for the purpose of scalability.

### Ricardian Contract

A Ricardian contract is a single contract document which is both easily readable by human and machines. It is formatted as a text file and digitally signed by the issuer of the contract.

The security of a Ricardian contract is achieved by OpenPGP and all the signing keys are included within the contract so eliminates the use of external authority or we can say that a Ricardian contract carries its own PKI with them.

### The Difference between Ricardian contract and Smart contract

The major difference between the smart contract and the Ricardian contracts is that Smart Contracts are executable code but Ricardian contracts are the agreements recorded in a single file and not executable on their own. A smart contract doesn't have to be a Ricardian contract and a Ricardian contract is not a Smart contract, but a Smart contract can execute a Ricardian contract.

### 51% Attack:

When a group of participants of the PDL, own more than 50% of the decision making power, or in other words, a group of participants forms an alliance and start rejecting transactions from other participants.

### Eternal Contracts:

The contracts which are active for infinite time.

### Coin

A coin has its own ledger and represent usually financial transactions.

### Token:

A token ususally represents an asset or some service and runs third-party ledger.

# Abbreviations

AML: Anti-Money Laundering

API: Application Programming Interface

SC: Smart Contract

CEN-CENELEC: European Committee for Standardization and European Committee for Electrotechnical Standardization.

DLT. Distributed Ledger Technology

EBP: European Blockchain Partnership

EBSI: European Blockchain Service Infrastructure.

EC: European Commission

EFTA: European Free Trade Association

eIDAS: Electronic Identification, Authentication and Trust Services.

EIRA: European Interoperability Reference Architecture

ESSIF: European Self Sovereign Identity Framework

ETSI: European Telecommunication Standards Institute

EU. European Union

FIG: International Federation of Surveyors

GDPR: General Data Protection Regulation

ICO: Initial Coin Offering

ICT: Information and Communications Technology

ISO: International Standards Organization

ITU: International Telecommunication Unit

KYC: Know Your Customer

OECD: Organization for Economic Co-operation and Development

PDL: Permissioned Distributed Ledger.

RA: Regulatory Authority

SG: Study Group.

SLA: Service Level Agreement

SME: Small and Medium Enterprise.

STO: Security Token Offering.

TOOP: The Once-only Principle

TSAG: Telecommunication Standardization Advisory Group.

UN/CEFACT: United Nations Centre for Trade Facilitation and Electronic Business.

UNCITRAL: United Nations Commission on International Trade Law.

UNE: Spanish Association for Standardization.

# Introduction to Smart Contracts

## Introduction

In general, an computer program deployed on a PDL can be considered a SC, but different schemes are possible to achieve this. In the next two sections, we will show the two main proposals adopted in practice, highlighting their similarities to classical programming languages paradigms.

## Smart Contract Programming Paradigms

Any PDL general goal is the distributed managing of a common data repository defining a current global state. There is no assumption on the type of data stored. When such data is executable code, then the induced global state can be seen as the state of a distributed virtual machine. Any executable code stored on a PDL is dubbed a “Smart Contract” (SC).

### Object-Oriented Paradigm

Historically, the main model adopted for SCs has been on the line of the traditional Object Oriented paradigm. As such, a SC is seen as a code entity composed of two main sections:

* Internal storage, in the form of identifiers – value associations akin to a dictionary, similarly to object fields;
* Functions’ definitions, specifying the set of actions allowed for the given SC with the appropriate scope modifiers, similarly to object methods.

A general contract definition, i.e. the writing of a SC code, is akin to defining a new class. A new SC can then be deployed to the PDL, akin to an object being instantiated from a class. Similarly to how an object has a reference, so a SC, once deployed, holds a unique identifier. The deploy operation of a new SC updates the global state, making all its fields and functions visible and callable, according to their respective visibility modifiers. Any accessible function of deployed SCs can then be called as many times as wanted, according to their operational logic specified by their code.

This model of SCs generally describes all PDL protocols derived from or inspired by the distributed consensus protocol Ethereum protocol, such as the Quorum protocol from JPMorgan[[1]](#footnote-2) , Tezos or stellar protocol etc.

### Properties

Smart contracts’ properties directly depend on the properties of the underlying PDL and some properties due to their .

EC – more details on properties- try to be more specific(A calls B and B calls A(recurrsive)).

#### Immutability

As any data on a PDL, a SC is immutable. This means that the code deployed on the PDL, specifying the contract logic, cannot be changed. The values contained inside a SC internal storage are mutable as expected. A consequence of Immutability is *Immortality*, since, once deployed, a SC cannot be removed. However, most PDL protocols allow for a logic marking of a SC as *disabled*, to disable future calls to a given SC. In order to disable a SC needs to specify an appropriate function inside its code.

#### Availability

An SC is always available as long as the underlying PDL is accessible. This means that a SC functions can be invoked, and fields can be read, as long as an entity has the appropriate privileges specified by the contract and PDL.

#### Transparency

Any entity, with the appropriate privileges, might inspect a SC code and current values. As such, it is transparent to all intended participants what the logic that the SC enforces is. Moreover, any call to a function of a contract is performed through a general state update on the PDL (i.e. transaction). As such, all function calls are recorded and traceable, providing *Auditability*. As a further remark, we observe how the code of the contract is itself auditable given a source code and a certain compiler, i.e. any user can re compile the source code to make sure that the object code actually executed on ledger corresponds.

#### Self-Execution

Any execution of a SC, i.e. an invocation to one of its visible functions, is performed by the PDL, not by the user invoking the SC, nor by the SC creator. The SC execution is protected by the distributed consensus of the PDL, as such, it is behind the control of any single party, and cannot be tampered with by them. This property induces the sub-properties of:

* *Atomicity*: a SC invocation runs entirely or fails without affecting the state;
* *Synchronicity*: a SC invocation is executed in a synchronous way;
* *Determinism*: a SC invocation returns the same result for any node executing it.

#### Reusability

SCs are coded once and executed multiple times. A given smart contract can be uses as template for a wider set of applications sharing the same high level logic. The actual behaviour of a given contract may change depending on the parameters which are set at invocation time. For example, the SC for cellular service is modelled with required fields for QoS metrics such as latency; all the operators, in this case, will be required to specify the latency they will provide.

## Storage

Smart contracts are stored in distributed ledgers; however, their storage depends upon the nature of ledger architecture. For example, in case of a public blockchain such as Ethereum, a smart contract will be stored by all nodes; on the contrary, in a permissioned blockchain such as Hyperledger, smart contracts are stored only on the nodes that are part of a given channel (an abstract point-to-point link between nodes) and are established through communication between nodes.

## Life cycle of a Smart Contract

A smart contract is a computer program, the difference is that the smart contracts are immutable, so it requires great care to program them and should be tested on several levels before deployment. This section presents the recommended life cycle a smart contract should follow in order to avoid the dangers of mis-programming.

A screenshot of a social media post

Description automatically generated

Figure 1: Life-cycle of a smart contract

# Smart Contracts – Planning, coding and Testing

## Introduction

Smart Contracts are software codes, and similar efforts are required to program them like any other software program. The difference is, however for any usual software, the bugs can be fixed in new releases or through software patches, but SCs stored in a DLT by design lack this provision. The deployed SC code becomes ossified to the system and further changes to the same code under the same identification is not permitted. Several researchers have identified instances, where a bug or loop-hole in the contract resulted in loss of tokens, for example, DAO attack costed $60 million and Ethereum had to create a hard fork to overcome the problem. Hence, the careful planning and scrutiny of the code before deployment to the ledger is of utmost importance.

### Planning Phase

A smart contract can be deployed in many ways and all the methods are dependent on its underlying ledger technology and should be acceptable if acceptable by the participants. Our goal here is a contract that can be trusted by two parties who do not trust each other. The planning of a smart contract will enable the participants to make decisions considering their priorities for example, if two companies willing to have run a business contract that should stay between them, they can have a side-chain with smart contracts deployed there. In the following paragraphs, we discuss two possible methods of deployment.

#### Ownership of a contract

Usually, the creator of a contract is the owner as well; the owner of the contracts have exclusive privileges such as contract destruction. However, in PDLs where contracts can be reused by several participants for several unrelated transactions, it is feasible to have a role-based ownership mechanism. In Role-Based ownership, the operations of a contract are governed by a group of participants with appropriate privileges; as PDL is a collaborative ledger, these privileges can be specific to a contract.

#### On-chain deployment

This is the simplest method for deployment of smart contracts and the contracts are stored directly in the master-ledger. The advantage is that the customers do not have to rely on any other side-chain and it is best for a centralized system. Since all the full contract codes are stored in the master-chain, in long-term scalability can be a problem.

#### Off-chain deployment

In this method, the main logic of a contract is stored in a side-chain and only address of that contract is stored in the master-chain. The advantage of this technique is that, since it is not required for a full-contract code to be in the master-chain, this technique is scalable. Additionally, the off-chain contract address in the master-chain can be updated by the owner of the contract with only the transaction. The danger in this type of deployment is that, if the off-chain contract is not self-destructive, it can stay forever and can be callable by other contracts, also as it is in the chain(no matter if the chain is side-chain) it occupies storage.

Another possibility of resuing off-chain smart contracts is that they can also be re-used by other new contracts, from the authorised contracts, but they must be handled with care.

A screenshot of a cell phone

Description automatically generated

Figure 2 On-chain and off-chain Smart contracts

There are options on how a smart contract lifecycle is managed after it has been initially deployed. Some of these mechanisms may be available natively in a specific ledger, and for other ledgers, this may require the use of programming techniques such as call delegation across contracts.

#### Immutable deployment

The simplest deployment model is where the smart contract is never terminated. In some ledgers, a smart contract can always be removed, while in other ledgers this decision can be built into the smart contract at development or deployment time (i.e. by choosing to include ot omit a “termination” mechanism).

#### Terminatable deployment

A smart contract may be terminated, i.e. permanently removed from the ledger, if the ledger or the smart contract itself directly supports this mechanism. Some ledger technologies, for example, support removal of a smart contract, but may require a *quorum* of different parties to agree to the removal before implementing it. For other ledgers, such as Ethereum, the contract termination must be built into the contract in advance, and cannot be retroactively added.

#### Upgradeable deployment

Some ledger technologies support upgrades to an existing smart contract, i.e. changing the smart contract’s operational code. This capability may be controlled by governance mechanisns, for example by requiring a quorum of different parties to agree to the upgrade. For other ledgers which view the contract itself as immutable, upgrades are managed by deployment of an interface contract (proxy contract), and of an implementation contract. The interface contract includes a mutable reference to the implementation contract. The interface contract acts as the single point of access to the smart contract, but it will only *delegate* all operations to the implementation contract. When an upgrade is performed, a new (immutable) contract is deployed, and the pointer in the interface contract is changed to point to the new implementation contract. The old implementation contract is later terminated.

#### Draft Template

In future, the smart contracting technology is envisioned to be used for every business transaction and this is the reason, efforts to bring standards for them from standardising bodies. Before the planning and coding process begins, a smart contract should be drafted electronically or manually. At this initial stage, all stockholders can decide together with their requirements such as code and resources requirements. The standardisation body recommendations should be considered in this step, as it facilitates the future steps of planning and coding.

#### Terms Negotiation:

Once the draft of requirements is ready, the terms and conditions between the parties should be decided and agreed. It is particularly important in the smart contracts as in the cases of traditional manual contract, there is a freedom of amendment at any time, whereas smart contract by-design does not have such freedom. At the same time, it is important that all parties agree on a single point after the entire deliberation so that there is no conflict in the future.

The terms and conditions will be varied from organisation to organisation and ledger to ledger but we outline some of the major points that can be part of the terms negotation:

1. Is the smart contract going to on-chain or off-chain?
2. If participants want to maintain a side-chain, who will be particpants and their role?
3. For how long the side-chain will be active?

A smart contract may be developed for the purpose of N-M interaction, i.e. one-to-one, one-to-many, many-to-one or many-to-many interactions. The smart contract, when deployed, is usually identified as being *governed* by a specific part (N=1) or a group of distinct parties. If the contract is governed by more than one party, a *consortium agreement* needs to be formulated within that group to outline the governance model that is applied to the smart contract. This agreement needs to take into account the legal and business aspects of the smart contract, and address issues such as who is eligible to stop, terminate, or upgrade the smart contract, and how these are enforced contractually or technically.

Especially in situations where contract can be stopped and resumed, terminated, or upgraded, the multi-party governance agreement must take into account who has the authority to issue these operations. If the ledger has immutable smart contracts, this governance model must also be encoded within the smart contract during the contract planning, as any changes later will be difficult to implement. For multi-party governance this requires decisions on the technical implementation aspects of:

* How are decisions communicated to the smart contract? Potential approaches are multisigs, where the group members individually sign a decision, which is then sent to the smart contract which verifies that the signatures are from the defined set of governing authorities, and that a sufficient number have signed it. Another option is to use voting, in which case an action is initiated, but the smart contract requires different parties to individually endorse the action (or reject it) within a time limit—each party sends its own endorsement or rejection separately to the smart contract.
* How are the governing parties recognized by the smart contract? Depending on the ledger, this may be an organizational identity within the ledger, or an account owned by the party, or a bare public key.
* Are actions allowed only by uninamous resolution, “N minus one” i.e. whole consortia minus one, “N out of M”, simple majority, or a more complex mechanism such as “at least one core member, and majority of everybody must endorse.” These must be defined accurately and encoded in the smart contract.
* What are the individual actions that the governance policy applies to (contract stop and resume, termination, contract upgrade, changes in governing party identities, and any other business-specific actions) and is the policy different for different actions?

Depending on the capabilities of the ledger itself some of these policy decisions may be part of the ledger itself; in other cases, these decisions must be encoded into the smart contract and defined in design phase already.

#### Compile Draft:

Afer the template draft(step 1-a) and negotiation of terms(1-b), the draft of the SC should be compiled. This step, can specify the complete supervisory level specifications such as, underlying ledger technology to be used, the stakeholder needs etc.

#### Review:

The last step of planning phase should be to review and verify the complete planning phase and check conditions such as if all the requirements are met. In this step, it should also be considered that if the contract is planned considering the standardisation body guildlines.

### Coding and Testing phase

#### Introduction:

As soon as the contract plan is in place, the next step is to code it. Only a complete and sophisticated planning step can produce a complete and error-free code. In this section, we will cover the coding and testing phase of a smart contract and discuss the steps which can help industries produce viabale contracts. A recommended testing flow is shown in Figure 3.

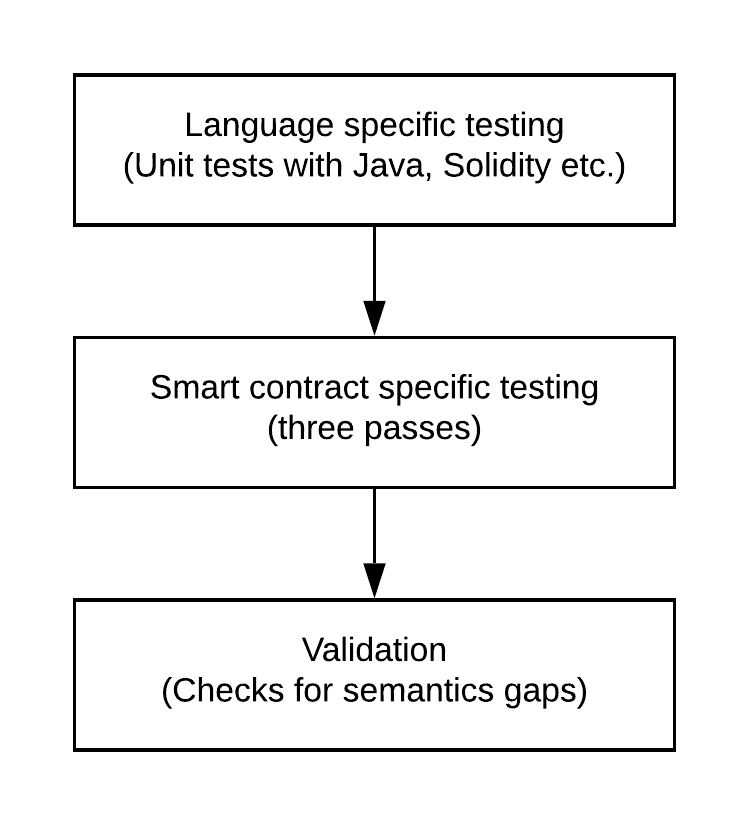


Figure 3: Smart Contract Testing process

#### Code/Programming language-level Testing

~~Refer to traditional software testing – mirror in the context of SC~~

The Smart Contracts’ testing varies from traditional software testing in several ways; In traditional software which is mostly meant to be coded for a limited number of users, and the executed on a single machine at a time. On the contrary, SCs is executed once and the distributed ledger comprising of several machines get updated with the results

#### Smart Contract specific Testing

##### Open Source SC Analysers

There are number of open-source SC analyzers such as Securify[[2]](#footnote-3) and SmartCheck[[3]](#footnote-4) are available to analyse the SC code and tag the vulnerabilities present in the program. These vulnerabilities such as unintiated functions, can provide third-party(possibly malicious) access to the contract, thus to the ledger. These analysers prevent external accesses by inspecting the code and flagging the possible vulnerablabilities in the code. However, all the analysers have their limitations such as they support certain ledger technology or certain programming language, also the attacks on the contracts are evolving, hence more comprehensive scrutiny of the code can be achieved by multiple analyses techniques. Another important consideration for analyser is the support for ledger technology, most of the available analysers are for Ethereum and Hyperledger and the adopters of the ledger should look for their own supported analyser.

##### Sandbox Testing

A PDL is a group of nodes, and errorounous smart contracts can be harmful to all of the nodes. A Sandbox testing mechanism is required before execution of a smart contract on PDL. These are specific to the ledger technology and a local copy of ledger can be used as a sandbox. Sample contracts should be run several times to ensure the desired level of security.

It is recommended that nodes run their pre-tests before sending the deployment transaction. These pre-tests are specific to the use-case and the technology. For example, in a token contract the address of the payee must be included in the contract and for the asset trail contract, the change of owner is an important parameter. Here, we highlight three reference passes for a contract, stakeholders should look for, before depolyment of a smart contract:

###### Three passes:

*Execution clauses:*

A contract is executed with certain predefined conditions which can be internal such as start time or external such as an API call. Hence, it is important to have clearly defined execution clauses in a contract, as its absence will make the contract dormant. Moreover, the presence of unintended conditions can open backdoors in a contract and should be avoided.

*Penetratable clauses:*

The clauses that invoke the critical parts of the contracts such as payment remittance should be accessed exclusively by the owner or the authorised member of the PDL. Moreover, all the entry points to the contract should be examined to ensure water-tight security.

*Termination caluses:*

Termination clauses allow the contract to stop its execution and become inactive; this prevents the ledger from having eternal contracts. Moreover, after a specific time, a contract should be self-destructible to avoid outdated versions of the contracts and allow the modified new versions.

Sandbox approaches also a useful set of tools e.g.: codius

Verification programming languages such as F by Microsoft

#### Validation

The smart contract should be the exact and true representation of natural language contract and should perform only the tasks specified in the natural language contract. In other words, semantic gap[Making SC smarter] that is the expected execution and the actual execution must be avoided in order to avoid the wrong doings of the contract and implement an error-free code.The sementics gaps should be checked at the Level 3 of testing process(Figure 3).

Consumer Protection

User-expereince exercise – (Ismael)

The most pertinent discussion here is that: does the smart contract truly and completely encode the contract logic? Is the software-code water-tight enough to prevent the unwanted execution of the contract by unauthorised parties?(to remove?)

### Deployment and Execution Phase

#### Deployment

Smart contracts by-design once deployed cannot be changed or amended, extensive emphasis has been placed on the earlier stages. In the deployment stage, the contract is installed on a PDL, and it particularly involves the stakeholders such as a mobile operator and a tractor vendor, who agreed on a contract for network services. This stage should not particularly involve the developers as the deployment can be straight forward if the earlier steps are carried out perfectly.

#### Execution

Deployed contracts can be theoretically executed unlimited times(depends upon the under-lying ledger architecture). The execution of a smart contract can be parameterized and non-parameterized depends on the design model and can be performed by any authorized party through an API.

TBD – comments from Ericsson -

#### Upgrading(discussed earlier – can be removed from here)

If the smart contract can be upgraded, either via the ledger’s native support (such as in Hyperledger Fabric, using versioned chaincode), or via development techniques (proxy contract), the process of upgrades needs to be managed. This may require communication with the users of the smart contract as with any software release management process. If the smart contract is governed by a group, they must coordinate the upgrade using the appropriate technical means.

#### Termination

A Smart contracts must be terminated exclusively or they should be self-destructable after certain time as they may contain critical conditions such as payouts. In this case, if a dormant contract exists in a ledger can be exploited by the adversary. The termination of the contract can be done by the contract itself(i.e. destroys itself) or through an API handled exclusively by the stakeholders through the digital-signature mechanism, to ensure security. The termination should exclusively be performed by the owner of the contract, and it is possible that instantiation of one contract terminates the older one.

# Architectural requirements for Smart Contracts

## Introduction

## Architectural requirements

### Resuability

Since a smart contract is a software that can live forever in a blockchain, its architecture should be able to provide flexibility for reusability of the contracts. The reusability can prevent the dormant contracts and the blockchain being populated. A method to implement is to use the dormant contract by other contracts.

### Self-Destruction

Some contracts are not such that they can be destroyed such as contracts with some monetary value can't be destroyed as that's the value which can only change owners but not destroyed. However, if the contracts are some kind of agreement, for example, an agreement between a user and their network service provider, it should include the self-destructive clause.

## Reference Architecture

### Introduction:

A smart contract should have three different layers, 1) Logic – in which the original purpose of a smart contract is defined, 2) Algorithm – the code logic and the interpretation of logic to steps of execution and 3) Code - the final code which must be a true representation of the Logic. In Figure xx, this layered architecuture is illustrated.

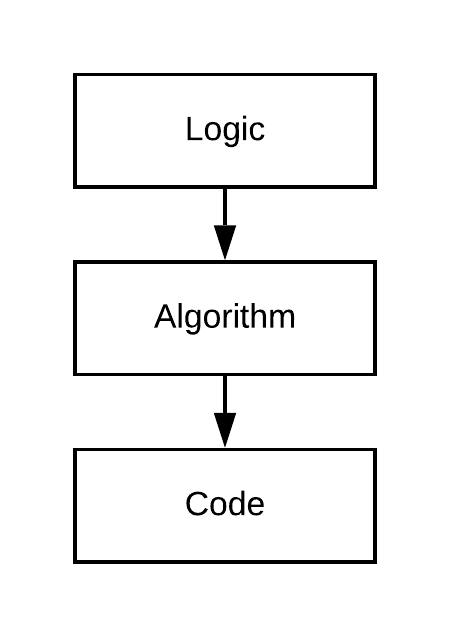


Figure 4:Layered process of a Smart contract

### Reference Architecture

The architecture explained here, provides the mechanism of resource allocation using smart contracts that can be adopted by industry for smart contract coupled resource allocation. Architecture figure below.

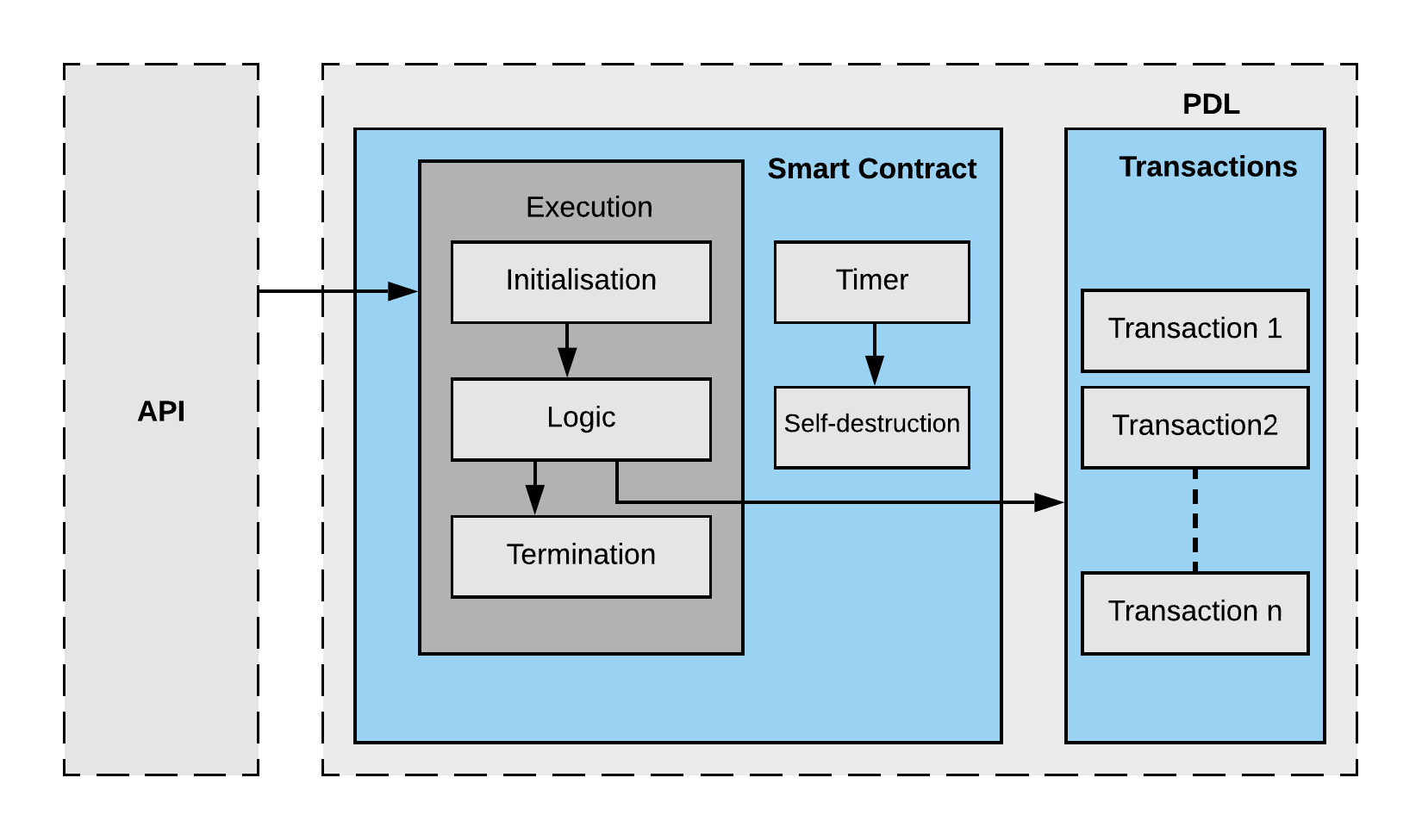


Figure 5: Reference Architecture of a Smart Contract

## Transaction Dependencies

It is most probable that a smart contract is dependent on other contracts, in this case, the order of execution of these contracts is of utmost importance. The contracts that are dependent on the transactions from other contracts cannot be executed in advance and must wait for their pre-requisites to be completed.

The transaction ordering for a smart contract must be defined in the consensus of the corresponding PDL. It is recommended to adopt specific ordering of transaction inside the base contract(i.e the contract which will initiate the chain of contracts) to avoid transactions being rejected and cause clutter in the ledger. With specific ordering all the dependencies must be clearly stated or referenced in the base contract that will access and all the transactions will follow the specified ordering.

# Smart Contracts – Applications, solutions and Needs

## Introduction

## Applications

## Solutions

### Introduction

### Scalabliity

#### Check-point

The side chains can self-destruct after a certain *check-point(for example 31st December)* and the hash of complete chain can be recorded in the master-ledger, to prove that there was a certain chain available there.

#### Extensibility

tbd

### Example: Smart contracts with QoS monitoring

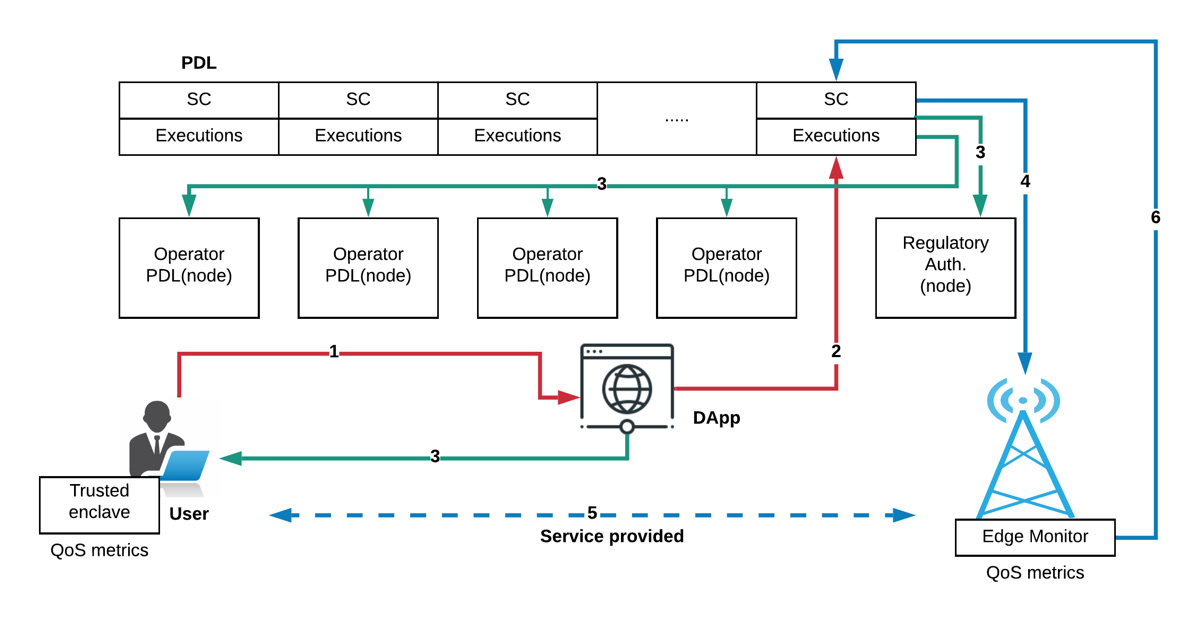


Figure 6: Smart contract with QoS monitoring

1. All the contracts from all available operators are advertised on a DApp where the customer can input their requirements and choose a suitable network service offer. The customer is required to forward the agreed upon payment to the chosen operator using traditional means.
2. Once a suitable offer is chosen, the web app fills an activation request to the corresponding contract, transferring the payment due at the same time (which will be actually carried only if the request is successful). This request is encoded as a transaction to then need to be signed by the customer, to prove their approval, and then sent to the PDL. Do note how all service offers are backed by already deployed SCs on the PDL to be able to accept requests. As such the SC deployment fees are paid by their operator owners, while activation requests fees are paid by the customers requesting them.
3. The new transaction containing the activation request is added to the pool of pending requests by the validators (i.e. operators and regulatory authorities), that will eventually accept it, through the distributed consensus algorithm, if well formed.
4. On successful execution, the respective operator gets notified and can start allocating the resource to provide the requested service.
5. The Service between the operator and the customer is being provided. At this stage, the actual QoS is managed by the Trusted enclave and the Edge Monitor.
6. The Edge Monitor records the receipts for the user back to the PDL inside the corresponding agreement SC. This allows to verify, and prove, if the SLA has been fulfilled.

## Needs - Requirements to build a viable system with Smart contracts

### Regulatory Aspects

### Security of the Contracts

Blockchains such as Ethereum, smart contracts are publically available; as per Ethereum consensus a copy of every contract is stored at every node; that is not so beneficial in many real-world applications, where all the participants, even from the same PDL are not involved in every agreement or contract.

In this situation, a more exclusive mechanism should be adopted, where only the involved participants, have access to the smart contracts. To ensure privacy in a smart contract, different access rights can be assigned to every participant of the contract. Here, the participants can be the direct trading parties or the other stakeholders such as the mediators.

### Enforceability

Smart contracts are self-executable, which means they can automatically execute with the fulfilment of a certain pre-coded condition;

~~this property makes them self-enforeceable but it doesn’t imply and practically theoretically eliminates the involvement of a third-party.~~When two or more parties internally or externally agree on a contract, they are expected to honour the agreement without any disputes and if there is any, the stakeholders can come together to resolve the issue as per organisation policies.

Legal disputes such as GDPR are beyond the scope of this report.

EC - technical discsussion is going towards legal. Technical specification is not replacing any law, it is pure techical nothing related to law. **Across border** – it is more policy issue than legal issue.

BL – smart contracts should address the problems of **GDPR**. Regulatory requirements are still applicable. smart contracts could help to sort out some of the issues. Germany might be doing something whichh is different than UK. Some of the problems can be solved through smart contracts.

### Availablity

As smart contracts are aiming to be adopted as a contract mechanism for industry, an important consideration for them is to be always available for execution.

Like other distributed systems, PDLs are vulnerable to attacks from malicious parties which can cause Denial of Service to legitimate users. Continuous and redundant service requests from malicious users can overwhelm the PDL and SC and must be prevented to make services available.

A global lock of certain time(possibly few seconds) should be applied to prevent such happenings.

### Attack

TBC CRs expected.

### Integrity

### Risks

In order for a smart contract to be deployed successfully, a certain number of the nodes must approve the deployment transaction. Now, there is a risk of 51% attack; where the majority of the nodes form an alliance start rejecting the legitimate transactions and since they have the majority it is likely that the contract will never be deployed. However, this risk in PDLs is low, as the member nodes are most of the time known to the consortium and admitted to the PDL after the authorisation. However, to propose a solution to this potential problem, in Figure 1, we introduce the regulatory authority view in which if the legitimate transactions are constantly being rejected, the regulatory authority can step-in and resolve the issue.

# Limitations of Smart Contracts

## Inter and Intra system threats

### Introduction:

ITU[[4]](#footnote-5) in its report on DLT and finanacial .. identified these potential risks to smart contract technology: *1) A reliance on a computer system itself that executes the contract, 2) flaws in the smart contract code( discussed in section 6) , 3) or the reliance on an external ‘off-chain’ event or person – to integrate with and execute – the embedded terms of the contract.*

### Absence of Termination clause/Self-Destruction

In every smart contract, a termination function is a fragile entity. If it does not exist or not programmed with the utmost care, can be active for an indefinite period, which can prove very dangerous. For example, if a contract is meant to be writing vehicle service records to the ledger such as location etc., and this car is sold by the company to another, the absence of or flaw in termination function can result in this vehicle to continue sending the critical data to the ledger. This is dangerous to the new owner of the car because his information, perhaps critical, is being seen by a third-party; also, for the old owner as this vehicle is still utilising the ledger and occupying the costly storageFor example, if a contract stipulates a payment for a certain period of time and the contract does not expire after that period, the amount will be paid indefinitely. Indeed, the payments can be cancelled by other means such as informing banks to stop the payment but that is alse dependent on the design of the contract. Moreover, if such errors go unnoticed, can potentially result in bigger losses.

Admission

### Accountability Control

Errorounous contracts if allowed in a PDL can kickoff transactions.

### Accountability Management of Smart Contract

PDL or SC?

### Poor Exception Handling

## Limitations

### Introduction

Most of the smart contracts' properties, if not understood, it can be a hindrance in their adoption for the industry. In this section, the inherit limitations of the smart contracts as a function of PDL are discussed:

### Occupancy:

One of the important points of a smart contract is that when they are installed on the ledger, they are immutable. It is also a well-off limit that is not optimal for everyone, as is often the case, as the contract contains deliberate provisions that are very important to change quickly. Not that smart contract is not possible to do this, but it is a costly task as it costs the resources to deploy and execute a contract, as well as every new contract blockchain lives on permanently. Over time as the volume of the blockchain increases, these unused contracts occupy space which can be better used by other contracts.

### Latency

The key consideration for deploying a smart contract is the delay or latency. The latency of a smart contract is the time it takes for a contract to get deployed and executed and can be categorised in 1) deployment latency and 2) execution latency.

Smart contracts get compiled on the local machines which can potentially be personal computers; then the request to deploy them is issued by the deployment entity through a transaction. In this situation, the smart contract latency is dependent on the compilation of the code and the network delay for a contract request to reach the chain.

Mostly, the Smart contracts get executed more often then deployment. the pre-deployed smart contract can be executed by any entity with the right permissions. To execute or invoke a smart contract a transaction is issued by the invoking entity and this depends upon the factors such as network connection and the congestion at the chain. Moreover, the nodes of the ledger by-design are distributed across the World and computation and speed limitations of every node adds an overhead to the latency in verification of contract transaction.

The method of deployment and execution discussed here is a high-level picture of the smart contract system and is strongly dependent on the underlying chain.

##### Underlying and Relying ledgers in permissioned context:

One of the most important considerations for the industry to adopt smart contract technology is that of the underlying ledger. Smart contracts are deployed on the ledger such as Corda, Ethereum or Hyperledger Fabric. Every ledger is unique in its properties and has different resource requirements. As of the time of writing this report, there is no system for ledgers to interact with different ledger exist, all the organizations or nodes must use the same underlying ledger technology in order to implement the smart contract as their contractual mechanism. This is not always possible for several reasons such as economically and feasibly to use same ledger technology; hence, be part of the consortium.

### Not every term can be translated to a Smart Contract(Remove this one?)

Smart contracts are nonetheless a computer program, and computer programs have very strict rules, such as if this then that or do this until this condition becomes true or false. But, in real-world contracts, the conditions are not always this rigid and there are leeways allowed intentionally by both parties, for example, if a business relationship between two organisations is old and they do want to give each other some discount but not to record in the contract, then it is difficult to have a smart contract in this situation. For a smart contract, either it is or it is not, there is no opportunity for a middle ground. Such biletral promoises which cannot be translated to the code, can be recorded in additional contract field in a plain text or in hash format, this will enable transparency between the participants. Adding this field in a hash form, can be verified later.

TBC: Need to explain Recursive.

### Legal Uncretainity:

PDL are distributed nodes, which can potentially be spreaded across the globe. The enforceability of smart contracts in different countries can be a issue.

|

# Annex A: Title of annex

Annex B:  
Title of annex

# B.1 First clause of the annex

## B.1.1 First subdivided clause of the annex

Annex C:  
Bibliography

Annex :  
Change History

| Date | Version | Information about changes |
| --- | --- | --- |
| 11-2019 | 0.0.1 | Initial draft – added table of contents |
| 12-2019 | 0.0.2 | Added SC introfuction, types and some text |
| 01-2019 | 0.0.3 | Cleaned draft |
| 02 - 2020 | 0.0.4 | Cleaned up after F2F meeting – deleted section 6 |
| 03- 2020 | 0.0.5 | Added life cycle + section 9 |
| 06-2020 | 0.0.6 | Limitations+cleared after F2F(online) meeting |
| 06-2020 | 0.0.6 | Added architecture |

# Document History

|  |  |  |
| --- | --- | --- |
| **Document history** | | |
| <Version> | <Date> | <Milestone> |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. https://docs.goquorum.com/en/latest/ [↑](#footnote-ref-2)
2. <https://dl.acm.org/doi/pdf/10.1145/3243734.3243780> [↑](#footnote-ref-3)
3. <https://dl.acm.org/doi/pdf/10.1145/3194113.3194115> [↑](#footnote-ref-4)
4. <https://www.itu.int/en/ITU-T/focusgroups/dfs/Documents/201703/ITU_FGDFS_Report-on-DLT-and-Financial-Inclusion.pdf> [↑](#footnote-ref-5)