ITUPublications
Recommendations

International Telecommunication Union
Standardization Sector

Recommendation

ITU-T F.751.12 (09/2023)

SERIES F: Non-telephone telecommunication services

Multimedia services

Formal verification framework for smart contract on distributed ledger technology

ITU-T F-SERIES RECOMMENDATIONS

Non-telephone telecommunication services

TELEGRAPH SERVICE	F.1-F.109
Operating methods for the international public telegram service	F.1-F.19
The gentex network	F.20-F.29
Message switching	F.30-F.39
The international telemessage service	F.40-F.58
The international telex service	F.59-F.89
Statistics and publications on international telegraph services	F.90-F.99
Scheduled and leased communication services	F.100-F.104
Phototelegraph service	F.105-F.109
MOBILE SERVICE	F.110-F.159
Mobile services and multidestination satellite services	F.110-F.159
TELEMATIC SERVICES	F.160-F.399
Public facsimile service	F.160-F.199
Teletex service	F.200-F.299
Videotex service	F.300-F.349
General provisions for telematic services	F.350-F.399
MESSAGE HANDLING SERVICES	F.400-F.499
DIRECTORY SERVICES	F.500-F.549
DOCUMENT COMMUNICATION	F.550-F.599
Document communication	F.550-F.579
Programming communication interfaces	F.580-F.599
DATA TRANSMISSION SERVICES	F.600-F.699
MULTIMEDIA SERVICES	**F.700-F.799**
ISDN SERVICES	F.800-F.849
UNIVERSAL PERSONAL TELECOMMUNICATION	F.850-F.899
ACCESSIBILITY AND HUMAN FACTORS	F.900-F.999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T F.751.12

Formal verification framework for smart contract on distributed ledger technology

Summary

Smart contracts can be used to reduce complex business contracts by directly enforcing the contract's payment methods and paybacks, and by automating the process of contract execution and verification into the network, without the intervention and cost of the person checking the contract's performance. However, smart contracts are a series of program codes generated on distributed ledger technology (DLT) and problems may occur in the process of executing the smart contract. As a method to solve problems that occur in the program execution environment, there is a formal verification. Recommendation ITU-T F.751.12 specifies the formal verification framework for smart contract on DLT, its overview, requirement and architecture in its framework, as well as the main technical direction of its formal method component. This Recommendation can be used as a guideline for a smart contract developer to build systems.

History [*]

Edition	Recommendation	Approval	Study Group	Unique ID
1.0	ITU-T F.751.12	2023-09-13	16	11.1002/1000/15626

Keywords

Distributed ledger technologies, security, smart contract, verification.

[*] To access the Recommendation, type the URL https://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID.

Table of Contents

Recommendation ITU-T F.751.12

Formal verification framework for smart contract on distributed ledger technology

1 Scope

This Recommendation provides a formal verification framework of smart contract on distributed ledger technology (DLT). It also includes:

– Overview of formal verification for smart contract;

– Requirements of formal verification for smart contract;

– Architecture of formal verification for smart contract.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T F.751.2]	Recommendation ITU-T F.751.2 (2020), *Reference framework for distributed ledger technologies*.
[ITU-T Y.3320]	Recommendation ITU-T Y.3320 (2014), *Requirements for applying formal methods to software-defined networking*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 consensus protocol [b-ITU-T X.1401]: Process for determining what blocks get added to the chain and what the current state is.

3.1.2 distributed ledger [b-ISO 22739]: Ledger that is shared across a set of DLT nodes and synchronized between the DLT nodes using a consensus mechanism.

3.1.3 smart contract [b-ITU-T F.751.0]: Program written on the distributed ledger system that encodes the rules for specific types of distributed ledger system transactions in a way that can be validated, and triggered by specific conditions.

3.2 Terms defined in this Recommendation

None.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

DApp	Decentralized Application
DLT	Distributed Ledger Technology

EV	Electric Vehicle

5 Conventions

In this Recommendation:

– The keywords "is required to" indicate a requirement which must be strictly followed and from which no deviation is permitted if conformance to this Recommendation is to be claimed.

– The keywords "is recommended" indicate a requirement which is recommended but which is not absolutely required. Thus, this requirement need not be present to claim conformance.

– The keywords "can optionally" indicate an optional requirement which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option and the feature can be optionally enabled by the network operator or service provider. Rather, it means the vendor may optionally provide the feature and still claim conformance with this Recommendation.

6 Overview of formal verification for smart contracts

6.1 Smart contract

Smart contracts are a series of program codes generated on the distributed ledger platform. It ensures that the contents of the contract are automatically executed on the assets contained in the distributed ledger when predetermined conditions are achieved. Smart contracts can be used to reduce complex business contracts by directly enforcing the contract's payment methods and paybacks, and by automating the process of contract execution and verification into the network, without the intervention and cost of the person checking the contract's performance. It is characterized by the ability to safely execute contracts based on low cost and agreement trust. Thus, it is possible to perform record storage, cash flow, and contract fulfilment on one platform, thereby increasing work efficiency.

Smart contracts reduce contract execution costs and the potential for disputes by automatically fulfilling contracts upon fulfilment of conditions. However, the cost of the contract writing phase for programming languages does not decrease. Since the contract is automatically executed, it gives the other party the confidence that the contract will be fulfilled. When combined with the disclosed transaction information, it reduces the risk of the transaction and reduces the uncertainty of the contract. It reduces the ambiguity of contract interpretation because it is written in program code. However, there is a possibility of inconsistencies in interpretation and writing errors in the writing phase when programming contracts in program languages.

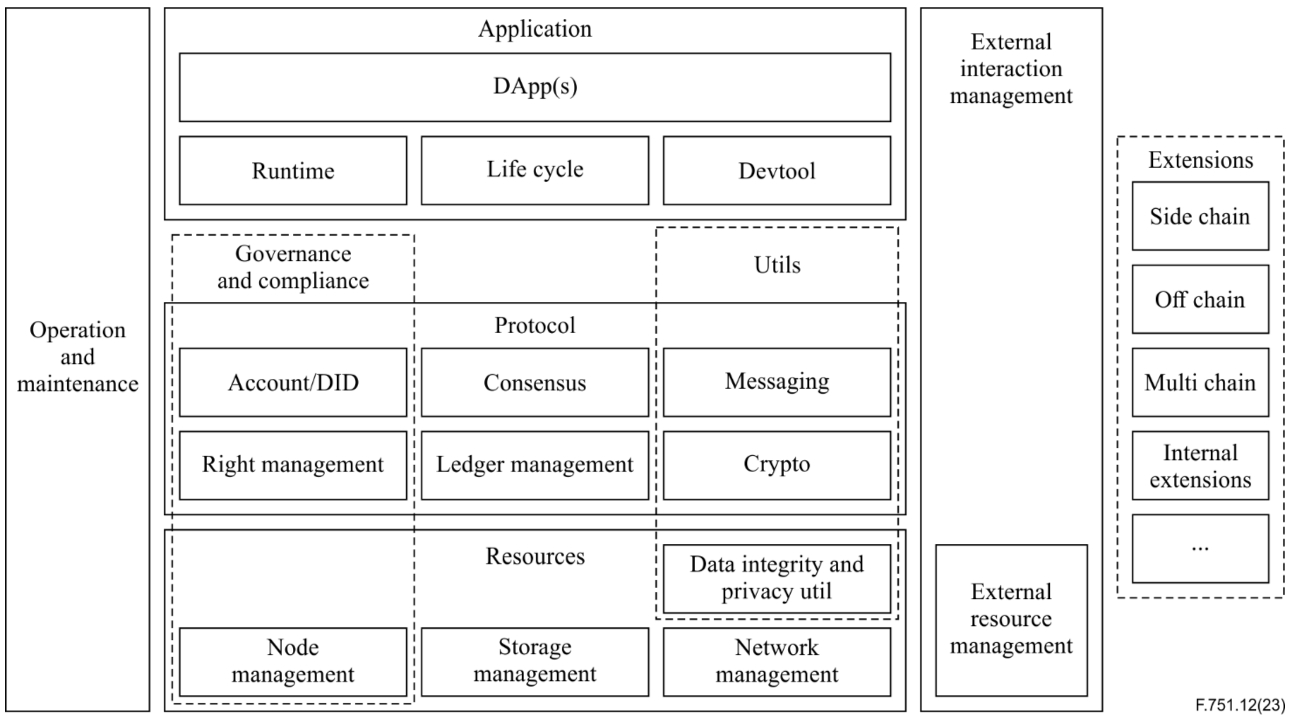


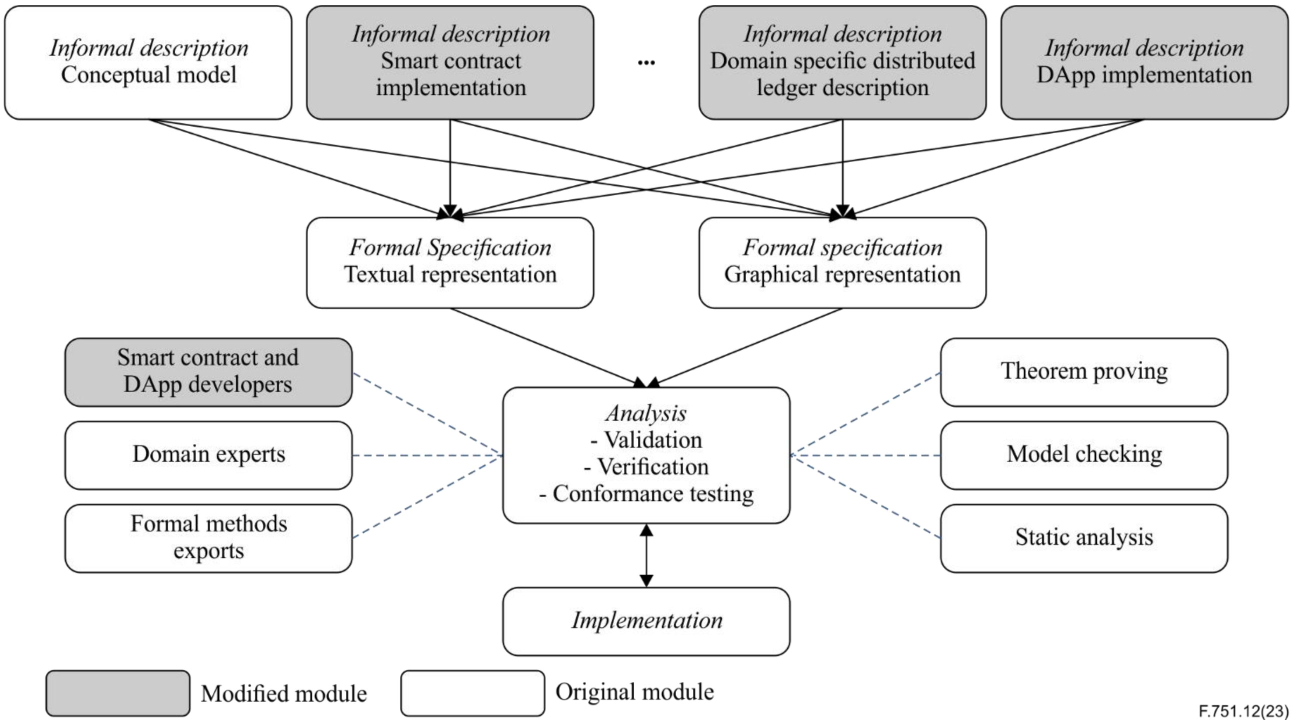
Figure 1 – High-level conceptual architecture of DLT (ITU-T F.751.2)

Figure 1 illustrates the high-level architecture constrains the highly abstract hierarchy of distributed ledger technologies. The smart contract mechanism is an optional in protocol component, but useful, component for DLT and most decentralized applications (DApps) are written by smart contracts. Smart contracts are programs written in a programming language that automatically execute contracts at specific times or events as defined by the parties. Therefore, smart contracts have the following problems.

– Since smart contracts are software code written in a programming language, they have the same vulnerabilities as overflow problems and exception handling problems. If, in the process of implementing the contract as a program, logic is omitted or wrong, there may be a vulnerability, or the scope of the variable may be specified incorrectly.
– Smart contracts have a property called contract, which causes dependency problems. In order for the contract to be confirmed as a block, a proof of work process is performed. In this proof of work, the contract order is changed, which causes a problem.
– Since smart contracts are recorded on the distributed ledger, it is difficult to apply patches when the smart contract is recorded on the distributed ledger. Since most smart contract developers create based on existing smart contract code, once the vulnerable code is raised, not only is it impossible to fix, but it also causes a problem that the error is expanded and reproduced.
– Vulnerabilities exist when writing smart contracts using data outside the distributed ledger. In the case of data outside the distributed ledger, it is difficult to check whether the data is forged or has been tampered with.

6.2 Formal method

Figure 2 is modified from [ITU-T Y.3320] and illustrates the overall flow from the informal description of the conceptual model, smart contract implementation and domain specific distributed ledger through to the implementation of a system using formal methods. Formal methods are those that specify or verify hardware/software systems in a way that is based on mathematics and logic. Characteristics to specify the system using mathematical symbols are also described using logical expressions to verify whether the system satisfies the specific conditions required by the user using mathematical properties to reduce the ambiguity and uncertainty inherent in natural language. In other words, it is possible to verify whether a complex system satisfies a specific condition and to have a reliable verified system.



Informal description Conceptual model
Informal description Smart contract implementation
...
Informal description Domain specific distributed ledger description
Informal description DApp implementation
Formal Specification Textual representation
Formal specification Graphical representation
Smart contract and DApp developers
Domain experts
Formal methods exports
Analysis - Validation - Verification - Conformance testing
Theorem proving
Model checking
Static analysis
Implementation
Modified module
Original module
F.751.12(23)

Figure 2 – Overall flow of formal methods

There are two types of formal methods: formal specification and formal verification. Formal specification is based on the environment in which the system will operate using formal logic or mathematical logic. It describes the requirements that the system must meet and the system design to fulfil the requirements. The formal specification can be further divided into requirements specification and design specification. A requirements specification is a specification that defines what the system must satisfy, and a design specification shows how the system is constructed. Formal verification is a technique that verifies the inconsistency and completeness of the system by analysing the formal specification, using the verification method provided by the formal logic or mathematical logic, or verifying whether the design satisfies the requirements in a given assumption. This describes methods that are working for verification of smart contract models.

– Theorem proving is that involves encoding a system and its desired properties into a particular mathematical logic. Then, it attempts to derive a formal proof of satisfaction of these properties based on the axioms and inference rules of the formal system.
– Model checking is a technique for automatically verifying a system model against its specification.
– Static analysis is any offline computation that inspects code and produces opinions about the code.

7 Requirements of formal verification for smart contracts

7.1 DApp requirements

This clause identifies a set of DApp requirements that must be supported by DLT.

– It is required that a DApp component provide an open interface in DLT. Through this, verification request and reply of smart contracts are performed.

– It is required that a DApp component provide the ability to build smart contracts and interact with the smart contract mechanism component.

7.2 Smart contract mechanism requirements

This clause identifies a set of smart contract mechanism requirements that must be supported by DLT.

– It is required that the smart contract mechanism component provide an open interface in DLT. Through this, the verification request and reply of smart contracts are performed.

– It is required that the smart contract mechanism component provide the ability to store a compiled execute file of the smart contract in the distributed ledger.

– It is required that the smart contract mechanism component provide the ability to store a related file of the smart contract in the distributed ledger.

7.3 Formal method requirements

This clause identifies a set of formal method requirements that must be supported by DLT.

– It is required that formal method component provide open interface in the distributed ledger environment. Through this, the verification request and reply of smart contracts are performed.

– It is recommended that formal method component provide verification function for script language problems of the smart contract, problems occurring in the process of interworking with internal or external data in distributed ledger and problems caused by a contract order problem.

– It is recommended that the formal method component allow access by authorized users.

– It is recommended that the formal method component provide formal syntax, interpret the meaning of scripting language and support distributed ledger external API and protocol interworking.

– It is recommended that the formal method component provide a function to check whether there are no logical errors between the existing contract in the distributed ledger and the smart contracts to be verified.

– It is recommended that the formal method component check whether there is an error in the order between smart contracts in the block to be created.

– The formal method component can optionally provide for verification of the user who composes or participates in a smart contract.

– The formal method component can optionally provide for verification of external data in a smart contract.

8 Architecture of formal verification for smart contracts

8.1 Overview

In the high-level conceptual architecture of DLT, smart contracts and formal method components are configured as shown in Figure 3, modified from [ITU-T F.751.2]. The high-level conceptual architecture of DLT requires smart contract and smart contract mechanism modules to build smart contracts. Formal method component for verification is also needed.

DApps are software that communicates with the DLT. The user interface of the distributed application is similar to a website or mobile application. The smart contract represents the core logic of a decentralized application. Therefore, these are executed through the interworking of smart contract mechanisms.

Smart contract mechanisms are integral building blocks of the distributed ledger that process information from external sensors or events and help the distributed ledger manage. By using smart contract mechanisms for event processing and expressing logic, many complex applications can be built on DLT. Smart contract mechanisms are used in the process of configuring and executing decentralized applications. However, some DLT can only be composed of a DLT without decentralized application. In this environment, smart contract writing and execution is done on the DLT.

A formal method component is configured to prevent errors in the process of writing and executing a smart contract. The formal method component has an interface to connect with the DLT and distributed applications. The formal method component is composed of a parser, a model builder and a model checker for smart contract and transaction data.

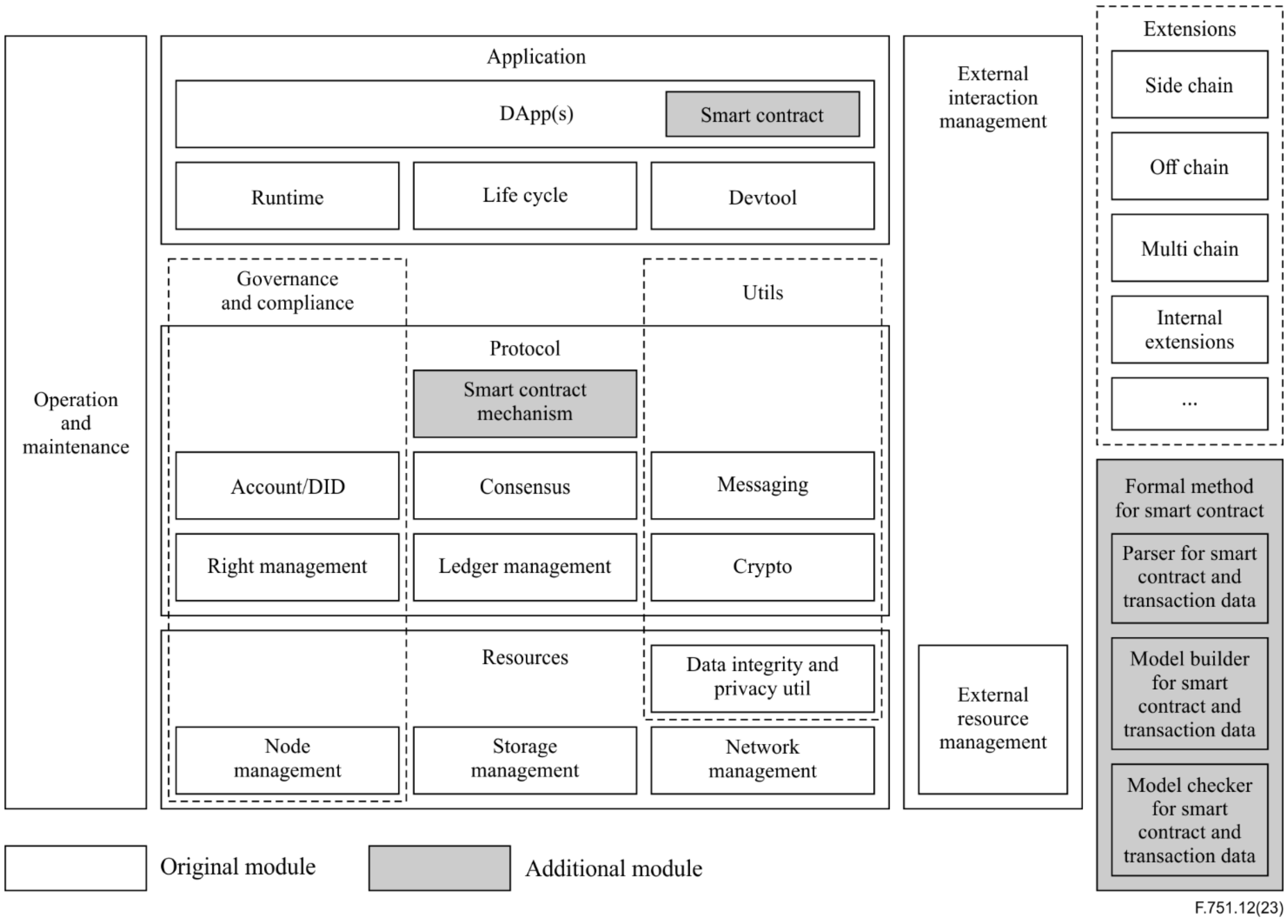


Figure 3 – High-level DLT architecture for formal verification

In order to provide formal verifications to the smart contract, additional functions comprised in the formal methods component are needed to process the parser, model builder and model checker for the smart contract. The parser for the smart contract function provides an open application interface for DApps or a smart contract mechanism to express syntactic analysis. The model checker for a smart contract function provides an open application interface for the DApp or smart contract mechanism to check whether new semantics can cause errors or operational conflicts within the DLT.

8.2 Formal verification

Smart contracts are expressed in various types of logic. They are composed of metadata and events constituting the contract. Problems can arise due to smart contracts built or participated in by malicious users. Therefore, previously written transactions and smart contracts are analysed to identify malicious users. Malicious user verification can be classified using reinforcement learning and deep learning methods. Based on this, it is possible to verify a user who composes or participates in a smart contract.

For smart contracts built or participated in by normal users, the verification application is used for verification. Verifying a smart contract consists of a formal verification in the form of graphical and textual representation. A process of transmitting and parsing related information through an external interface is required. It has an architecture in which a verification model is created and verified based on parsed information. The relationship between the verification module and the smart contract for this is shown in Figure 4.

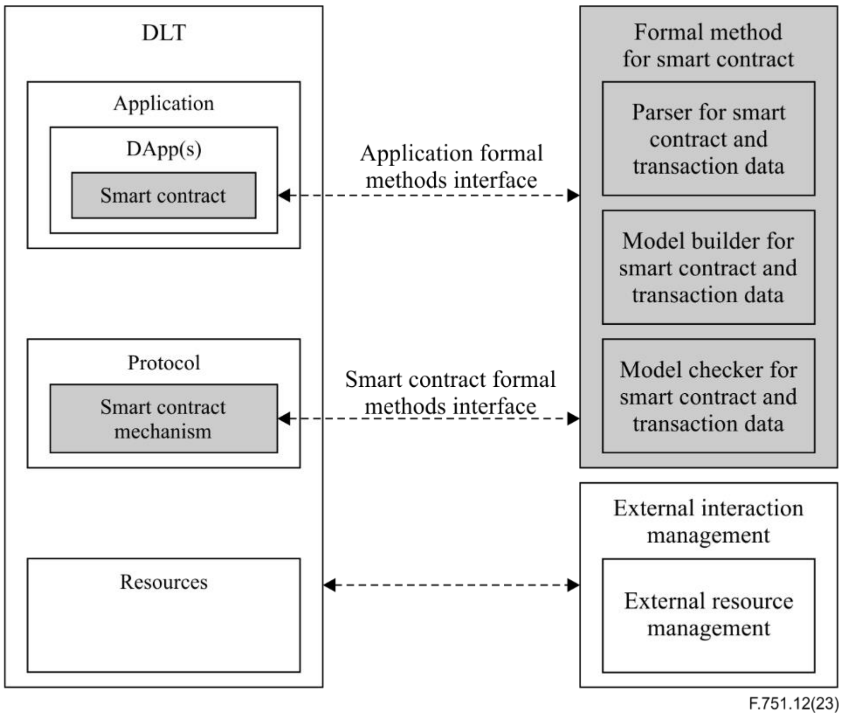

DLT
Application
DApp(s)
Smart contract
Protocol
Smart contract mechanism
Resources
Application formal methods interface
Smart contract formal methods interface
Formal method for smart contract
Parser for smart contract and transaction data
Model builder for smart contract and transaction data
Model checker for smart contract and transaction data
External interaction management
External resource management
F.751.12(23)

Figure 4 – Relationship between the formal methods component and the DLT

Formal verification is performed in the following procedure:

– The smart contract or smart contract mechanism module requests verification to the formal method component.

– The formal method component requests and receives smart contract code, transaction data, and external data.

– The parser module parses smart contract and transaction data.

– The parser module verifies whether any of the smart contract author and participants is a malicious user.

– The parser module verifies the validity of external data.

– The model builder module creates a formal verification model based on the parsed data.

– The model checker performs verification based on the created model and replies with the result.

Data used to build smart contracts in DLT can be divided into external data and internal data. Internal data refers to data managed by DLT. External data is data provided from outside DLT. In the case of internal data in DLT, the data can be verified during the smart contract verification process.

In order to use external data in a smart contract, a smart contract for external data verification or a provision site is required. Through this, external data is authenticated and accessed. A flow chart for this is shown in Figure 5. When a smart contract execution request is received from a user, the smart contract checks whether external data is used. If external data is used, it checks whether there is a related data verification smart contract and executes it. External data verification is performed in the procedure shown in Figure 5.

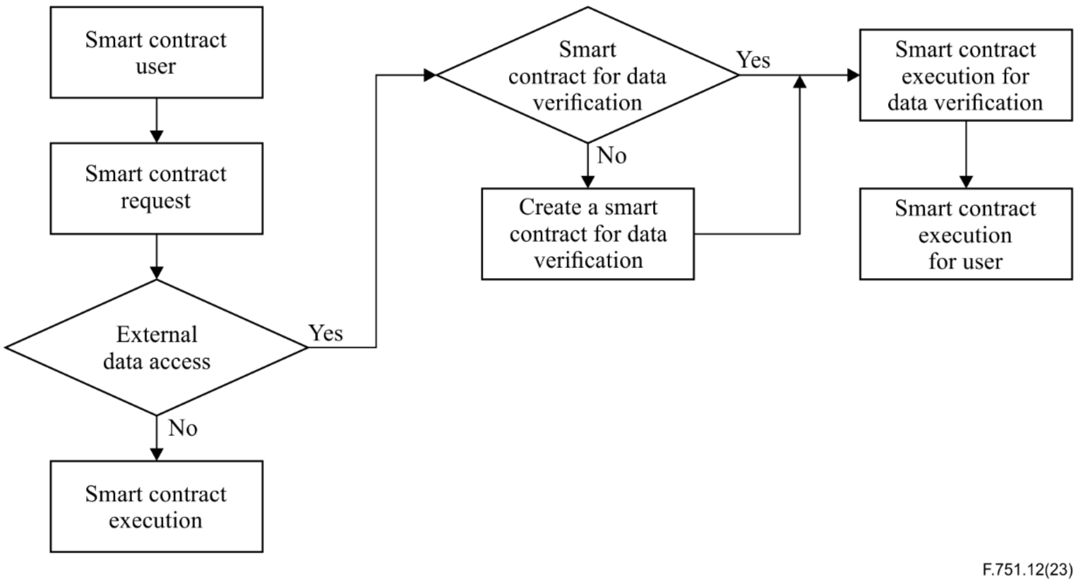

Smart contract user
Smart contract request
External data access
Yes
No
Smart contract execution
Smart contract for data verification
Yes
No
Create a smart contract for data verification
Smart contract execution for data verification
Smart contract execution for user
F.751.12(23)

Figure 5 – Flow chart for smart contract based external data verification

The procedure is as follows:

– The user requests smart contract execution in DLT.

– The system checks whether external data is used.

– The system checks that a smart contract has been created to verify the use of external data.

– The system executes smart contracts for external data usage and smart contracts requested by users.

Appendix I

Use case of verification for smart contract

(This appendix does not form an integral part of this Recommendation.)

I.1 Overall procedure for formal verification

A smart contract is written using the program language supported in the distributed ledger and stored in the ledger in the form of an executable file. A smart contract is executed according to the event conditions. There are two ways to write these smart contracts, through a smart contract module or through a DApp. The procedure for writing and verifying smart contracts through a DApp is as follows.

– Building a smart contract through DApp.
– The smart contract written by DApp is delivered to the formal method component.
– The parser classifies smart contracts written in a programming language.
– A model builder generates a verification model from the parsed smart contract.
– The model checker is used to verify, and the results are delivered to the DApp.
– The DApp decides whether to execute a smart contract based on the verification results.
– The smart contract module is applied to the DApp according to whether the contract is executed or not.

I.2 Internet of Things (IoT) environment

Various distributed ledger platforms have emerged for the Internet of Things (IoT) environment. In addition, various distributed ledger applications that support the IoT environment have been developed and applied to services. In this distributed ledger platform and application for the IoT environment, smart contracts that support events between IoT devices are very important.

Table I.1 – Distributed ledger platforms or applications for IoT

Platforms	Distributed ledger for IoT, which overcomes the inefficiencies of current distributed ledger designs and introduces a new way of reaching consensus in a decentralized peer-to-peer system
	A consortium exploring the role distributed ledger may have in providing security for the IoT.
	A decentralized network for IoT powered by a privacy-centric distributed ledger
Applications	E-charging mobility: Share & Charge was a "distributed-ledger-based mobility product for EV users, allowing users to share charging stations across a decentralized platform without middlemen."
	Energy: Combining IoT devices (smart meters) with DLT simplified the process of tracking and supplying renewable energy to end users
	Wireless network: A decentralized machine network simplifies connecting anything to the Internet through a distributed ledger, wireless network and open-source software
	Location data sharing: Provides infrastructure for the transparent and compliant exchange of location data; foot traffic sensor readings, store visit information and commute routes

I.2.1 Smart contract verification for resource sharing service

Resource sharing services support IoT device resource sharing, such as for bicycles. Consumer, providers and service providers are required to provide resource sharing services. Figure I.1 shows a distributed-ledger-technology-based resource sharing platform.

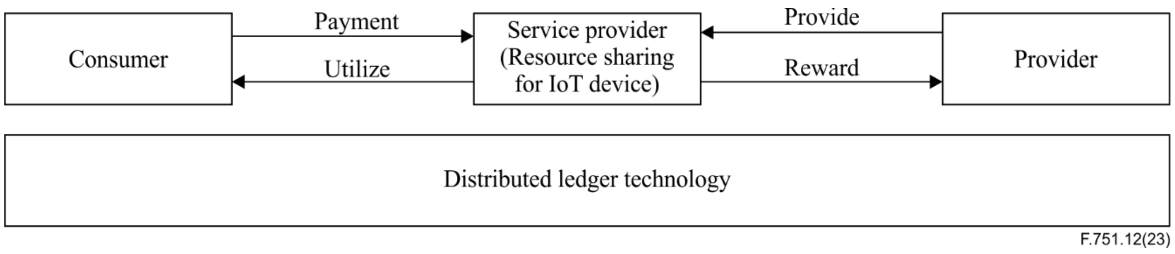

Consumer
Payment
Utilize
Service provider (Resource sharing for IoT device)
Provide
Reward
Provider
Distributed ledger technology
F.751.12(23)

Figure I.1 – DLT based resource sharing service

The consumer builds a smart contract and requests a service to utilize the resource sharing service. It is necessary to check if there is an error in the smart contract written by the consumer, so verification is performed before a smart contract is executed. The overall process for this is shown in Figure I.2.

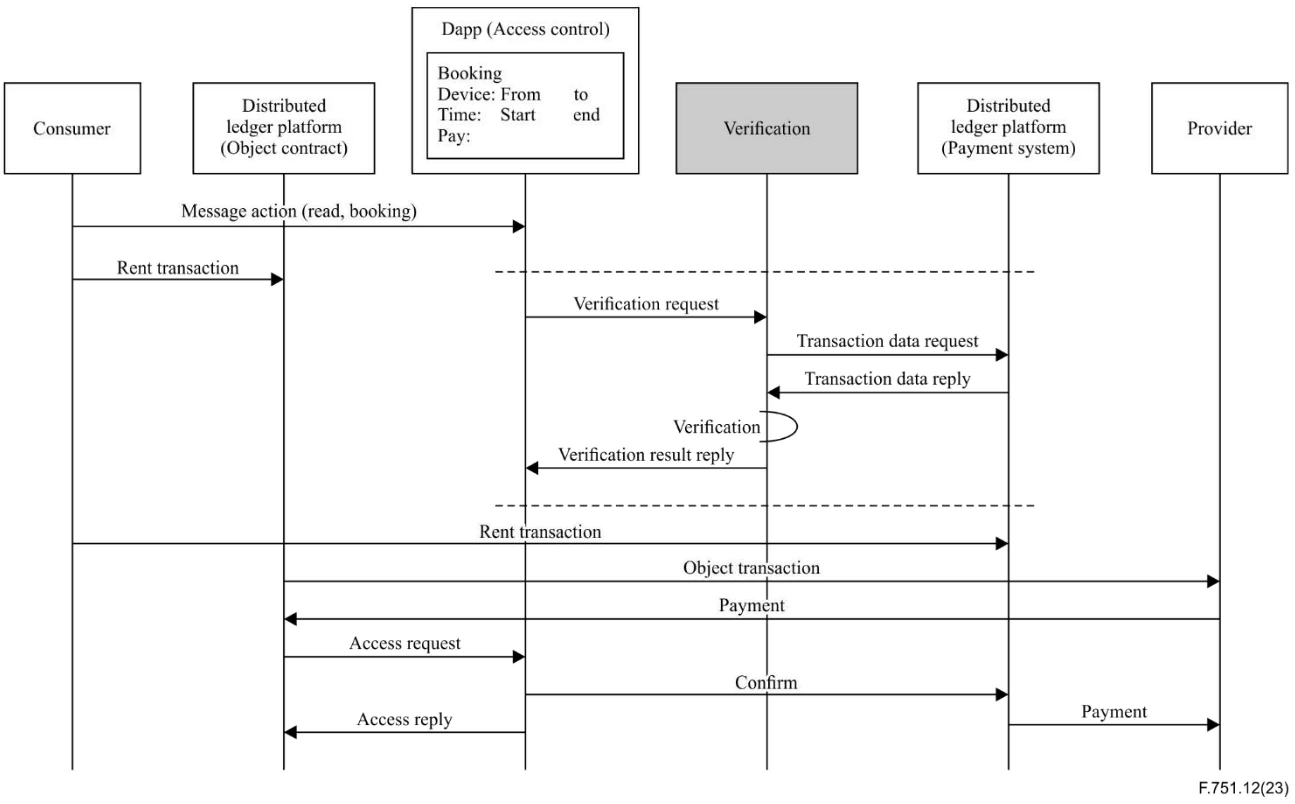


Figure I.2 – Overall process of smart contract verification for resource sharing service

Appendix II

Related projects on verification for smart contract

(This appendix does not form an integral part of this Recommendation.)

VeriDis: Verification of Distributed Systems

– This project aims to exploit and further develop the advances and integration of interactive and automated theorem proving applied to the area of concurrent and distributed systems. The goal of this project is to assist algorithm and system designers to carry out formally proved developments, where proofs of relevant properties as well as bugs can be found completely automatically.

– Reference site: https://team.inria.fr/veridis/publications/

Certik: Building Fully Trustworthy SmartContracts and distributed ledger Ecosystems

– This platform is envisaged to be a formal verification framework for building fully trustworthy smart contacts and distributed ledger ecosystems. Different from the traditional testing approaches to detect bugs, this platform attempts to mathematically prove that distributed ledger ecosystems are bug-free.

– Reference site: https://certik.org/

Bibliography

[b-ITU-T F.751.0]	Recommendation ITU-T F.751.0 (2020), *Requirements for distributed ledger systems*.
[b-ITU-T X.1401]	Recommendation ITU-T X.1401 (2019), *Security threats to distributed ledger technology*.
[b-ISO 22739]	ISO 22739:2020, *Blockchain and distributed ledger technologies — Vocabulary*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	**Non-telephone telecommunication services**
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2023