CM-009

ブロックチェーンシステムでの不正データ混入防止に向けたスマートコントラクト自動生成ツールの開発
Smart Contract Generator:
Preventing Inappropriate Data Contamination in Blockchain System

津田 奈子 †　　山田 将史 †
Nako Tsuda　Masafumi Yamada

1　はじめに

ブロックチェーン (BC) 技術は、やり取りされる情報の信頼性が分散ネットワーク内の合意形成のプロセスによって担保され、かつ、改ざんなどの不正を系全体で防ぐことができることから、トレーサビリティの担保を目的としたシステムへの活用が検討されている [1]。

BC は、データの改ざんが困難であり、登録されたデータに対する信頼性は高い。しかし、BC が信頼性を担保するデータは、登録後のデータに対してであるため、登録するデータの信頼性は担保できない [2]。そのため、データ登録前に、ルールに則った登録データであることを検証するなど、登録データの正当性を担保し、不正データの混入を防止する必要がある。

ルールに則った登録データであることを担保する方法の一つとして、BC システムへの入力前に外部システムで確認する方法が挙げられる。しかし、外部システムが不正アクセスによって改ざんされていないことや、不当なデータ検証でないことを、BC システムでは担保できない。そのため、BC システム内でデータを検証し、ルールに則った登録データであることを確認した上で、データを登録することが考えられる。BC 基盤上で動作するデータ制御プログラムであるスマートコントラクト (SC) は、BC システム内でデータを検証することに適しているが、機能が限られているため、各ルールをハードコードした多数の SC を実装する必要がある。

そこで、本研究では、登録データを検証するスマートコントラクトの自動生成ツール (SC 自動生成ツール) と、ルールに基づく登録データ検証の実施を保証するデータコントロールスマートコントラクト (データコントロール SC) を用いて、登録するデータの正当性を担保する手法を考案した。

2　BC システムでの不正データ混入防止に向けた提案手法

SC 自動生成ツールとデータコントロール SC を用いた提案手法は、図 1 のような BC システムである。提案手法は、登録データを検証する SC(データ検証 SC) の生成、データ登録、データ参照の機能を持つ。

2.1　データ検証 SC の生成・デプロイ

提案手法でのデータ検証 SC の生成およびデプロイについて説明する。提案手法では、検証ルールを、BC ノードで SC として動作する SC 自動生成ツールに送信し、データ検証 SC を生成する。この際、データ検証 SC が SC 自動生成ツールから生成された SC であることの証明のため、SC 自動生成ツールがデータ検証 SC をデプロイする。使用する BC に SC が SC をデプロイする仕組みがない場合は、生成した SC のデータのハッシュ値を分散型台帳に登録し、別途データ検証 SC をデプロイする。(図 1：生成①〜③)

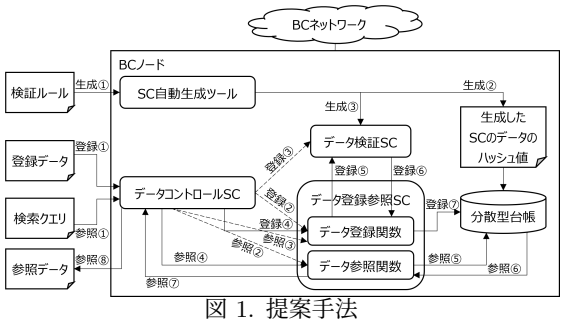

図 1. 提案手法

2.2　データ登録

提案手法でのデータの登録について説明する。データの登録は、まず、使用するデータ登録参照 SC のデータ登録関数を指定して、登録データをデータコントロール SC に送信する。次に、データコントロール SC では、使用するデータ登録参照 SC のデータ登録関数が、データ検証 SC を呼び出していること、検証結果をもとに登録可否を判断していること、および、データ登録がデータコントロール SC からのみ実施できることを確認する。さらに、使用しているデータ検証 SC が、BC ノード内の SC 自動生成ツールから生成された SC であることを、デプロイ元アドレスが SC 自動生成ツールであるか、または、台帳に登録された SC のデータのハッシュ値と一致するかによって確認する。データ登録関数とデータ検証 SC の正当性が確認された場合は、登録データをデータ登録関数に送信する。最後に、データ登録関数は、データ検証 SC による検証を実施し、検証ルールに則ったデータである場合は、台帳に登録する。(図 1：登録①〜⑦)

2.3　データ参照

提案手法でのデータの参照について説明する。データの参照は、まず、使用するデータ登録参照 SC のデータ参照関数を指定して、検索クエリをデータコントロール SC に送信する。次に、データコントロール SC では、使用するデータ参照関数のデータ登録参照 SC 内で対となってい

†三菱電機株式会社 情報技術総合研究所, Information Technology R&D Center, Mitsubishi Electric Corporation

るデータ登録関数の正当性、および、データ参照がデータコントロール SC からのみ実施できることを確認する。データ登録関数の確認手順は、データ登録時と同様である。データ登録関数とデータ参照関数の正当性が確認された場合は、検索クエリをデータ参照関数に送信する。データ参照関数は、検索に該当するデータを台帳から受信し、受信したデータをデータコントロール SC に送信する。最後に、データコントロール SC は、検索結果を返す。(図 1：参照①〜⑧)

2.4 提案手法の効果

SC 自動生成ツールを使用することで、検証ルールの記述以外に人の手を介さず SC が生成できるため、SC へのバグの混入をなくし、不正データを混入させるようなデータ検証を防ぐことができる。また、検証ルールの記述のみでデータ検証 SC を生成できるため、複数のデータ検証 SC の作成、および、当該 SC の妥当性確認にかかるコストが削減できる。

しかし、SC 自動生成ツールの使用のみでは、データを登録するための SC で、SC 自動生成ツールから生成したデータ検証 SC を使用することは担保できていない。そのため、データ登録の際に、本来行われるべき検証がなされないままデータが登録され、不正データが混入する可能性がある。

そこで、データコントロール SC を用いることで、データ登録参照 SC のデータ登録関数およびデータ検証 SC を検証した上でのデータ登録実施を保証し、実施すべき検証を行った正当性のあるデータが登録されたことを担保できる。また、データ参照時にはデータ参照関数を用いて取得したデータが、正当なデータ登録関数によって登録されたデータであることを検証することで、正当性のあるデータを参照していることを担保できる。

3 提案手法の実現に向けた SC 自動生成ツールの開発

3.1 概要

本研究では、BC OSS の一つである Hyperledger Fabric[*1] (HLF) を活用した、製品のトレーサビリティの担保を目的とするシステム (トレーサビリティシステム) に、提案手法を組み込むことを検討している。

別の BC OSS である Ethereum[*2]では、Ethereum の SC の開発言語である Solidity[*3]の一機能として、SC が SC を生成する機能を紹介している [3]。一方で、HLF の SC(本稿では HLF の chaincode と同義とする) の開発言語である go 言語は、BC に特化した開発言語ではなく、SC が SC を生成する機能を持たないため、何らかの手法で SC 自動生成ツールとなる go 言語プログラムを作成する必要がある。

[*1]Hyperledger Fabric は、The Linux Foundation の商標です。
[*2]Ethereum は、Stiftung Ethereum (Foundation Ethereum) の登録商標です。
[*3]Solidity は、Stiftung Ethereum (Foundation Ethereum) の登録商標です。

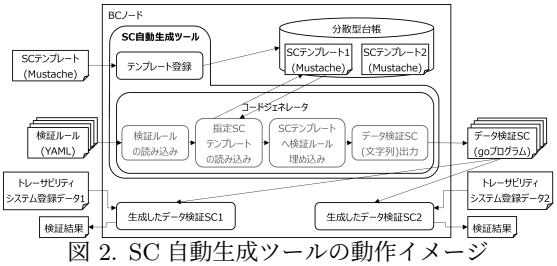

図 2. SC 自動生成ツールの動作イメージ

そこで、本研究では、提案手法の実現に向けて、YAML と Mustache を使用した go 言語のメタプログラミング手法 [4] を用いて、SC として動作する SC 自動生成ツールを開発した。

3.2 SC 自動生成ツール

SC 自動生成ツールは、BC ノード内で SC として動作し、YAML 形式で記載した検証ルールを、Mustache を用いて記載した SC テンプレートに埋め込むことで、go 言語で記載されたデータ検証 SC を出力するツールである。SC 自動生成ツールの動作イメージは、図 2 の通りである。

SC 自動生成ツールは、テンプレート登録機能と、コードジェネレータ機能から成る。コードジェネレータを使用するためには、テンプレート登録から、SC テンプレートを登録する必要がある。SC テンプレートは、Mustache を用いて記載した、生成するデータ検証 SC の基となるテンプレートであり、検証の要件に応じて作成する。また、コードジェネレータを使用する際には、YAML 形式で記載した検証ルールを作成する。検証ルールは、SC テンプレートに埋め込むデータであり、SC テンプレートの記載内容に合わせた記述とする。検証ルールを SC 自動生成ツールのコードジェネレータに送信すると、コードジェネレータでは、検証ルールの読み込み、指定の SC テンプレートの読み込みの後、SC テンプレートへ検証ルールを埋め込み、文字列としてデータ検証 SC を出力する。HLF では、SC が SC をデプロイする仕組みを持たないため、データ検証 SC を BC ノード内で動作させるためには、生成されたデータ検証 SC を、外部からデプロイする必要がある。

SC テンプレートと検証ルールの具体例を説明する。製品のデータで検証が求められる場合、SC テンプレートでは、製造番号の正規表現を用いて製造番号の表記が正しいかを検証する機能を持つテンプレートを作成する。検証ルールでは、製造番号の正規表現を記載する。型番それぞれに応じた検証ルールが存在する場合、型番に応じた検証ルールを作成し、型番に応じたデータ検証 SC が生成できる。これにより、型番ごとにデータ検証 SC を記述する必要がなくなり、プログラミングに馴染みがなくとも扱いやすい YAML 形式で検証ルールを記載するのみで、型番に応じたデータ検証 SC を生成することができ、開発が効率化できる。

4　SC 自動生成ツールの機能検証

機能検証では、データ検証 SC が生成できること、生成したデータ検証 SC がデプロイできること、デプロイしたデータ検証 SC が設定した検証ルールに則した検証結果を出力できることを確認する。

4.1　検証環境

SC 自動生成ツールのデプロイ環境として、Linux[*4] (ubuntu[*5] 18.04) の OS 上に、バージョン 2.2.6 の HLF を構築する。HLF のネットワーク構成は、HLF が提供している fabric-samples[5] の test-network を使用する。ネットワークの生成および SC のデプロイは、fabric-samples で用意されているシェルスクリプトを使用する。

4.2　機能検証に使用する SC テンプレート

SC 自動生成ツールを使用するためには、SC テンプレートを作成する必要がある。製品のトレーサビリティシステムへ組み込むことから、本研究では、「製品のデータ検証」の SC テンプレートを作成し、機能検証する。

製品のトレーサビリティシステムにおいて、製品のデータは、製造番号・ロット番号、数量、品名、型式、製造元、製造年月日、トレーサビリティ ID、登録日時、登録者を管理している。

「製品のデータ検証」では、以下 7 個の内容が確認できる SC テンプレートを作成する。製品のデータ検証では、製造元と型式によって、検証ルールとして指定する内容が異なることを想定し、以下の内容とした。また、検証ルールによって指定する部分は、Mustache の Variables を用いて、SC 自動生成ツールが検証ルールを埋め込めるよう SC テンプレートを作成する。検証結果は、製品のデータそれぞれの項目を map のキーとして、検証ルールを満たす項目は「0」、満たさない項目は「1」を map の値とするような構造体である。

1. 製造番号またはロット番号のどちらで管理されているか (ルール名：category)
2. 製造番号またはロット番号が指定の文字数であること (ルール名：numberlength)
3. 製造番号またはロット番号が指定の正規表現にマッチする文字列であること (ルール名：numberrule)
4. 製品の数量の下限と上限を満たす数量であること (ロット管理の製品を考慮)(ルール名:quantitymin、quamtitymax)
5. 型番に応じた指定の品名であること (ルール名：productname)
6. 製造年月日と登録日時が指定の年月日以降であること (ルール名：productdate)
7. 指定の登録者が登録していること (ルール名:account)

[*4]Linux は、Linus Torvalds の登録商標です。
[*5]ubuntu は、Canonical Ltd. の登録商標です。

表 1. 検証ルール 1

No.	ルール名	値
1	category	sernumber
2	numberlength	7
3	numberrule	[0-9][0-9][1-9XYZ][0-9]{4}
4	quantitymin quamtitymax	1 1
5	productname	製品 A
6	productdate	1285858800000
7	account	user1

表 2. 検証ルール 2

No.	ルール名	値
1	category	lotnumber
2	numberlength	3
3	numberrule	[0-9][0-9][1-9XYZ]
4	quantitymin quamtitymax	5 10
5	productname	オプションバッテリ
6	productdate	1285858800000
7	account	user2、user3、user4

4.3　機能検証に使用する検証ルール

機能検証のために用意した検証ルールを、表 1、表 2 に示す。これらの検証ルールは、機能検証のために作成したルールである。なお、ルール名 category における sernumber は製造番号管理、lotnumber はロット番号管理を示す。

4.4　検証方法

まず、第 4.1 節で用意した環境に、SC 自動生成ツールをデプロイする。次に、SC 自動生成ツールのテンプレート登録機能を用いて、第 4.2 節の SC テンプレートを分散型台帳に登録する。その後、SC 自動生成ツールのコードジェネレータ機能を用いて、第 4.3 節の検証ルール 2 個からデータ検証 SC2 個を生成する。データ検証 SC が生成できた場合、生成されたデータ検証 SC を HLF へのデプロイにあわせて整形してパッケージ化し、デプロイする。データ検証 SC がデプロイできた場合、表 3、表 4、表 5 の製品データを入力し、データ検証 SC からの出力が、設定した検証ルールに則した検証結果となっているかを確認する。なお、表中の「(空)」は空の文字列の入力を示す。

4.5　検証結果

表 1、表 2 の検証ルールを入力として、SC 自動生成ツールのコードジェネレータ機能を用いて、データ検証 SC が出力できることを確認した。また、出力したデータ検証 SC がデプロイできることも確認した。表 3 の製品データを、表 1、表 2 から生成したデータ検証 SC に入力した結果、および、表 4 の製品データを、表 1、表 2 から生成し

表 3. 製品データ 1

No.	データ項目	値
1	製造番号	2210001
2	ロット番号	(空)
3	数量	1
4	品名	製品 A
5	製造年月日 (UnixTime ms)	1642172400000
6	登録日時 (UnixTime ms)	1642209300000
7	登録者	user1

表 4. 製品データ 2

No.	データ項目	値
1	製造番号	(空)
2	ロット番号	221
3	数量	5
4	品名	オプションバッテリ
5	製造年月日 (UnixTime ms)	1642172400000
6	登録日時 (UnixTime ms)	1642209300000
7	登録者	user3

表 5. 製品データ 3

No.	データ項目	値
1	製造番号	AA10001
2	ロット番号	2201
3	数量	11
4	品名	製品 B
5	製造年月日 (UnixTime ms)	1283858800000
6	登録日時 (UnixTime ms)	1283858800000
7	登録者	user5

たデータ検証 SC に入力した結果、表 5 の製品データを、表 1、表 2 から生成したデータ検証 SC に入力した結果は、表 6 の通りである。これにより、表 1、表 2 の検証ルールから生成したデータ検証 SC が、設定した検証ルールに則した検証結果を出力することを確認した。

また、検証ルールを YAML で記載した場合の行数と、生成された SC の行数は表 7 の通りである。

4.6 考察

第 4.5 節の検証結果より、SC 自動生成ツールを用いて、設定した検証ルールに則した検証結果を出力できるようなデータ検証 SC が生成できることが確認できた。そのため、SC テンプレート 1 個の妥当性の確認のみで、データ検証 SC をそれぞれ複数人が作成するよりも信頼性の高いデータ検証 SC が生成できることがわかった。また、表 7 の結果より、それぞれデータ検証 SC を作成する場合と比べ、YAML の記述のみでデータ検証 SC が生成できることから、約 9 割の記述量削減が実現でき、開発効率化できることがわかった。さらに、開発中のトレーサビリティシステムは、複数事業者が運用し、それぞれの事業者が検証ルールを検討し、データ検証 SC を開発することが想定されているが、SC 自動生成ツールを使用することで、開発したデータ検証 SC の正当性を逐次系全体でカバーする必要がなくなり、データ検証 SC の開発コストも削減できるため、トレーサビリティシステム導入のハードルを下げることが期待できる。

一方で、HLF では、SC が SC をデプロイする仕組みを持たないため、生成されたデータ検証 SC を外部からデプロイする必要があり、その際に、生成されたデータ検証 SC をデプロイにあわせた形に整形し、パッケージ化するという行程が発生する。当該行程において、不正やバグが混入する可能性がある。実運用に向けては、当該機能を持つようなツールの開発、または、HLF でのシステム構築を前提として、SC 自動生成ツールに代わるような機能および構成の検討が必要である。

表 6. 検証結果

	製品データ項目 No.		1	2	3	4	5	6	7
製品データ 1	検証ルール 1	想定	0	0	0	0	0	0	0
		結果	0	0	0	0	0	0	0
	検証ルール 2	想定	1	1	1	1	0	0	1
		結果	1	1	1	1	0	0	1
製品データ 2	検証ルール 1	想定	1	1	1	1	0	0	1
		結果	1	1	1	1	0	0	1
	検証ルール 2	想定	0	0	0	0	0	0	0
		結果	0	0	0	0	0	0	0
製品データ 3	検証ルール 1	想定	1	1	1	1	1	1	1
		結果	1	1	1	1	1	1	1
	検証ルール 2	想定	1	1	1	1	1	1	1
		結果	1	1	1	1	1	1	1

表 7. SC 生成結果

	YAML の行数	SC の行数
検証ルール 1	10	115
検証ルール 2	12	117

5 おわりに

本研究では、BC システムに登録するデータの正当性を担保するため、データコントロール SC と SC 自動生成ツールを用いた手法を考案し、提案手法の実現に向け、SC 自動生成ツールを開発した。結果として、SC 自動生成ツールを用いて、設定した検証ルールに則った検証結果を出力するデータ検証 SC が生成できることを確認した。また、SC 自動生成ツールによって、データ検証 SC の開発効率化ができることも確認した。

今後は、提案手法の実現に向けて、データコントロール SC の開発を進める。また、HLF における提案手法の実運用に向け、SC 自動生成ツールから生成したデータ検証 SC をデプロイする機能を持つツールの開発、または、SC 自動生成ツールに代わるような機能および構成の検討を進める。

参考文献

[1] Kentaroh Toyoda, P Takis Mathiopoulos, Iwao Sasase, and Tomoaki Ohtsuki. A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain. *IEEE access*, Vol. 5, pp. 17465–17477, 2017.

[2] 宮原進. ブロックチェーンの信頼の仕組みと限界. https://kpmg.com/jp/ja/home/insights/2019/01/blockchain-system-structure.html, 2019. (Accessed on 2023/04/20).

[3] Ethereum. Contracts — solidity 0.5.4 ドキュメント. https://solidity-jp.readthedocs.io/ja/latest/contracts.html, 2019. (Accessed on 2023/04/20).

[4] 古川俊太. YAML+Mustache+go-generate で go のメタプログラミングをする. https://qiita.com/shunta-furukawa/items/ab1eac26cf5eb9348be9, 2020. (Accessed on 2023/05/11).

[5] Hyperledger. GitHub - hyperledger/fabric-samples. https://github.com/hyperledger/fabric-samples, 2023. (Accessed on 2023/05/10).