「マルチメディア，分散，協調とモバイル
(DICOMO2017) シンポジウム」 平成29年6月

ブロックチェーンを用いた持続可能な商品販売システムの検討

安井 瑛男[1]　空閑 洋平[2]　中村 修[1]

概要：本論文では、商品取引における販売者の役割を在庫の維持に限定し、商品交換および在庫の管理をパブリック型ブロックチェーンを用いて自動化することで、商品取引において高い持続可能性を実現する手法を提案する。

Toward a Blockchain-based Retail Kiosk for Sustainability

Akio Yasui[1]　Yohei Kuga[2]　Osamu Nakamura[1]

1. 本論文の概要

本手法は、商品販売用端末と利用者識別用端末、およびブロックチェーン上に構築された在庫管理機能により構成される。商品販売用端末と利用者識別用端末はそれぞれ内部に Ethereum ノードを持つ。在庫管理機能は Ethereum ブロックチェーン上に実装され、利用者による商品情報の追加、あるいは商品の購入を記録する。ブロックチェーンに記録された情報は、商品販売用の端末と全ての利用者識別用端末間で共有される。

一般的な商品販売における形態 (従来型販売形態) では、販売者が商品交換の監視および在庫管理の両方を行う。一方で、野菜の直売所に代表される無人販売所や、江崎グリコ社が提供するオフィスグリコ[*1]のようなサービスにおける形態 (無人型販売形態) では、販売者から利用者に対する信頼に基づき商品交換の監視を不要とすることで、商品販売者による取引の管理を簡略化している。しかしこうした形態においても、一定期間ごとに販売者が在庫の確認と売上実績の集計、および在庫の補填を行う必要がある。

商品交換の管理および在庫の管理について、その両方の責務を販売者が負うことは、継続的な商品販売を阻害する要因となりかねない。そこで本研究では、利用者に対する信頼に基づき商品交換の監視を省略し、さらにブロックチェーン上を用いて取引の記録および在庫管理を自動化し、販売者の責務を在庫を維持するための商品の追加に限定することで、商品販売における高い持続性を実現する。

[1] 慶應義塾大学環境情報学部
Keio University Faculty of Environment and Information Studies
[2] 慶応義塾大学院政策・メディア研究科
Keio University Graduate School of Media and Governance

[*1] https://www.glico.com/jp/enjoy/service/officeglico/

2. 関連技術

2.1 Everledger

Everledger[1] は、ブロックチェーンを用いて商品の流通経路の記録及び証明を行うことで、品質の証明と詐欺や盗難のリスクを低減できるとしている。最初の応用例はダイアモンドで、これまでに百万個以上のダイアモンドが登録されている。

2.2 Blockai

Blockai[2] は、アーティストが作品の情報をブロックチェーンに書き込むことで所有権を主張するためのサービスを提供している。Blockai は Web ページから作品の画像をアップロードすることで利用できる他、Twitter や Instagram と連携し、投稿した画像を自動的にブロックチェーンに書き込むことができるサービスも提供している。

3. Ethereum

3.1 概要

Ethereum[3] は、Ether を単位とする暗号通貨であると

同時に、ブロックチェーンを一つのステートマシンと捉えた分散コンピューティングプラットフォームである。

Ethereum では Ethereum Virtual Machine (EVM) と呼ばれる仮想マシンとその命令セットが定義されている。Ethereum におけるコントラクト・アカウントは、この命令セットで記述された EVM コードと、マシンの状態を表すストレージを持つ。EVM コードは命令ごとに消費する Ether 量 (gas) が決まっており、EVM コードを持つアカウントに対する送金は手数料を必要とする。この手数料は、EVM コードの実行と実行結果の検証の報酬として検証者 (miner) に支払われる。

3.2 Solidity

Solidity[4] は Ethereum プラットフォームにおけるコントラクト記述用言語であり、EVM コードにコンパイルされる。

Solidity ではコントラクトを、構造体とそこに定義されるメンバ関数として記述する。また、`modifier` と呼ばれる処理の共通化機構を持ち、関数が実行される条件や実行後に必ず行われる処理を構造化して記述できる。

Solidity は静的型付け言語であり、`bool`、`int`、`string` といった基本的な型の他に、Ethereum アカウントのアドレスを表す `address` や、`bytes1` から `bytes32` までの固定長バイト列といった型が用意されている。

4. 提案設計

4.1 概要

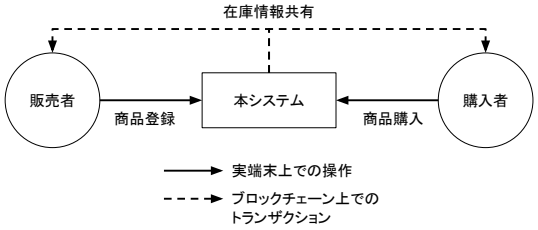

図 1 システムの概要

本提案システムは、全ての利用者が参加者となるブロックチェーン上に構築される。図 1 は、提案手法の概要を示す。

本システムにおける商品取引は、商品情報の追加 (商品登録) および商品の購入 (商品購入) からなる。商品登録および商品購入の履歴は全てブロックチェーンに書き込まれ、全利用者間で共有される。

以下に商品登録および商品購入のそれぞれについて設計を示す。

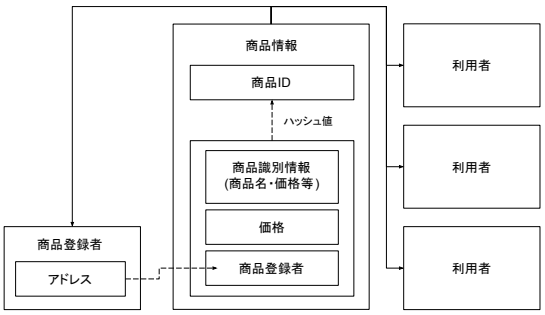

図 2 商品登録のフロー

4.2 商品登録

図 2 は、商品登録におけるフローを示す。

各商品は、商品の種類、価格、商品登録者ごとに一意である必要がある。そのため、商品の ID としてそれらの情報のハッシュ値を用いる。また商品の種類は、商品識別情報により区別される。商品識別情報とは、商品名や商品の画像といった利用者が購入時に商品を識別するための情報である。商品登録者は商品登録時に商品識別情報と価格を任意に設定し、本システムに登録する。その際、商品情報に商品登録者のアドレスが加えられた上で、商品識別情報、価格、商品登録者のハッシュ値を商品 ID として付与する。

4.3 商品購入

商品購入者は商品識別情報をもとに購入する商品を選択し、商品 ID とその件数を指定して購入する。本システムは商品購入者のアドレスから商品に紐付けられた商品登録者のアドレスに送金する。

5. 実装

本研究では、提案システムにおける商品管理機能および商品取引機能を、Ethereum 上のスマートコントラクトとして実装した。実装には Solidity を用いた。以下にスマートコントラクトの実装を示す。

5.1 仮通貨

実装を簡単にするため、取引には Ethereum における仮想通貨である Ether ではなく、スマートコントラクト内で管理される独自通貨 (仮通貨) を用いる。

ブロックチェーンに参加する任意の利用者はスマートコントラクトをデプロイすることで商品販売を管理するノード (ショップ) となる。ショップは他利用者に対する仮通貨の発行権を持つ。

5.2 構造体の設計

提案システムは、在庫データと各利用者の仮通貨保有量を持つコントラクトとして表現される。図 3 に、コントラクトの設計を示す。

```
contract Shop {
  address public shop;
  mapping (bytes32 => Item) public items;
  mapping (address => uint) public wallets;

  function ShopDelta() {
    shop = msg.sender;
  }
}
```

図 3 店舗を表すコントラクト

shop はスマートコントラクトをデプロイした利用者のアドレスを指し、仮通貨の発行などショップとなった利用者のみしか行えない処理の実行を制限するために用いる。このコントラクトがデプロイされた際に実行されるコンストラクタ関数にて設定される。

items は商品情報および在庫情報の管理に用いる。商品 ID ごとの商品情報へのアクセスを効率化するため、商品情報のリストは商品 ID をキーとした連想配列として持つ。商品を表す構造体の設計は図 4 に示す。

wallets は利用者ごとの保有仮通貨量を表す、利用者のアドレスをキーとした連想配列である。

```
struct Item {
  string name;
  uint price;
  address seller;
  uint amount;
}
```

図 4 商品を表す構造体

図 4 は商品を表す構造体である。name は商品識別情報の一例として商品名を格納する。price は本システムにおける仮通貨によって示される商品の価格を表す。seller は商品登録者のアドレスである。amount は現在システムに当該商品の在庫件数を表す。

5.3 仮通貨発行処理の実装

まず、ショップとなった利用者以外による処理の実行の制限を簡便にするために、図 5 のような modifier を定義する。

```
modifier onlyShop() {
  if (msg.sender != shop) throw;
  _;
}
```

図 5 処理の実行をショップとなった利用者に制限する modifier

次にこの modifier を用いて、ショップとなっている利用者だけが行える他利用者への仮通貨発行処理を定義する。

```
function mint(
    address user,
    uint amount
  ) onlyShop {
    wallets[user] += amount;
}
```

図 6 仮通貨発行処理

5.4 商品登録処理の実装

```
function sell(
    string name,
    uint price,
    uint amount
  ) returns (bytes32 id) {
  var seller = msg.sender;
  id = sha3(name, price, seller);
  var item = Item(name, price, seller, amount);
  items[id] = item;
}
```

図 7 商品登録処理

図 7 に商品登録処理の実装を示す。商品登録処理は商品識別情報 (ここでは名前)、設定する価格、追加する在庫件数を引数とし、登録された商品 ID を返す関数として定義される。この関数は引数として渡された情報から SHA-3 ハッシュ関数を用いて商品 ID を生成し、商品情報のリストに追加する。

5.5 商品購入処理の実装

図 8 に商品購入処理の実装を示す。商品購入処理は商品 ID と購入する件数を引数とする関数として定義される。要求された件数が在庫件数より多いか、また商品購入者の保有仮通貨量が商品価格よりも多いかを確認した上で、商品の在庫件数を減らし、商品購入者と当該商品登録者の保有仮通貨量をそれぞれ変更する。

6. 結論と今後の展望

Ethereum 上のスマートコントラクトとして提案システムを実装することで、信頼性を担保した形で商品情報と在庫の管理、および商品取引の管理を自動化することができた。

今後の展望として、本システムの持続可能性を示す上で必要な実装を整理する。現状、スマートコントラクトとして実装された本システムが提供しているインターフェースは Ethereum ノードを介した RPC のみであるため、別途

```
function buy(bytes32 id, uint amount) {
  var buyer = msg.sender;
  var item = items[id];

  var stock = item.amount;
  if (stock < amount) throw;

  var balance = wallets[buyer];
  if (balance < item.price) throw;

  item.amount -= amount;
  items[id] = item;

  wallets[buyer] -= item.price;
  wallets[item.seller] += item.price;
}
```

図 8 商品購入処理

利用者が簡便に利用できるインターフェースが必要である。具体的には各利用者が Ethereum ノードを動作させるためのクライアント端末およびソフトウェアを実装し、端末同士を接続することで商品の登録および購入処理に必要なアドレスの交換を行えるようなクライアントを想定している。また、現状では商品登録者および価格が異なる商品は別物として扱われているため、利用者のクライアント側で別途商品識別情報を用いて整理し、購入時は自動的に価格の安いものから選択していくといった工夫が必要である。クライアント端末およびソフトウェアの実装を行った上で本システムを運用し、利用状況を評価することで持続可能性の実現性を検討したい。

参考文献

[1] Everledger Ltd.: A Digital Global Ledger, Everledger Ltd. (online), available from ⟨https://www.everledger.io/⟩ (accessed 2017-03-01).

[2] Blockai: Blockai: Register your copyright and monitor how it's used on the web, Blockai (online), available from ⟨https://blockai.com/⟩ (accessed 2017-03-01).

[3] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger, *Ethereum Project Yellow Paper*, Vol. 151 (2014).

[4] Ethereum Foundation: The Solidity Contract-Oriented Programming Language, Ethereum Foundation (online), available from ⟨https://github.com/ethereum/solidity⟩ (accessed 2017-03-01).