Omi

ウェブアプリケーション分権的に稼働する

鹿目 黎 ’20 10/17

この資料の位置付け

- なにがわかるか　→　Omiの**概要**。細かくいうと、設計思想と、大雑把な設計と、Omiはなにが美味しいか、なにがまずいか。
- なぜかくか　→　Omiを**俯瞰的に紹介**したい。いろんな人に自分の思想を知ってもらい、**コメントをもらいたい**。

あくまで俯瞰的な紹介です。もし詳細が気になる方は最後にGithubのリンクを貼ってますのでご参照ください。

もくじ

- 最初に問題意識の共有や現状について記述する。
- 0miの大雑把な紹介。0miを構成するレイヤと、アーキテクチャの概観。
- なぜヘンテコなアーキテクチャになるのか記述。
- 課題を記述。論文(笑)を公開しているGithubのリンクを記載。

0miの紹介に入るまえに
問題意識の共有

Web2では利用者以外の管理者や開発者といった主体が、

- 利用者の個人情報をみる (不透明性)
- 得た個人情報を不正利用する (不透明性)
- 利用者の意図しない操作を行う (検証不可能性)
- アプリの利用を禁止する (検閲可能性)

ことができる。それを踏まえ、Web3ムーブメントが進行中…

Omiの紹介に入るまえに
Web3ムーブメントとはなにか

- ウェブアプリに検閲耐性と透明性、ウェブアプリのデータに検証可能性を持たせる動き。ウェブアプリにそれらの特質を持たせることで、ウェブ全体をより良く民主的なものにしようとしている。

- 現状は？→ ブロックチェインを中心として進んでいる。例えばPolkadotやDFinityなど、多くのプロジェクトがある。

※ DFInityはブロックチェインを使用しているというより、ガバナンストークンによる支配を行なっているという表現の方が適切かも

Omiの紹介に入るまえに

検証可能性、検閲耐性、透明性とはなにか　（曖昧さ回避）

- 通常のWeb3の文脈では検証可能性は、「受け取ったデータの内容が本当なのかどうかわかる」ということ。
- 検閲耐性は、国や企業などの権力者によって特定の人物のサービスの利用が制限されないということ。
- 透明性は、「アプリに挿入した自分のデータがどのように利用されるか」や「どのようなデータが取得されているか」などについて利用者がわかるということ。

次からOmiの紹介にはいる

0miの位置づけ

- 現状のDAppsと呼ばれるものは大抵、ユーザー認証など、ウェブアプリに必要な機能は集権的。0miはウェブアプリに必要な機能をアプリ開発者やホスティングマシーンのインセンティブを設計しつつ、分権的に稼働させる。
- DFinityはトークンでシステムを支配している。DFinityはあくまでプラットフォーム。0miはプラットフォームではなく、プロトコルの集合体。というか、ある種の考え方のようなもの？

  プラットフォーム化しないことで、今後の自由な発展を望める。

Omiの基本的な考え方(Concept) 1コ目

確かに、金融や契約などの高信頼性が要求されるアプリケーションには検証可能性が必要となる。しかし検証可能性が全てのウェブアプリケーションに必要とは言えない。そこで、検証可能性が必要になった場合にのみ外部のブロックチェインを使用する。世界中のコンピュータが同じデータを持つ必要が減るため、経済的合理性が増すと考えた。

ウェブアプリケーションに必ず備わっていなければならない機能は「受け取ったデータが本人からか」という(データではなく)送信者についての検証可能性。前にTwitterでソーシャルエンジニアリングを利用したハッキングが起こったが、Twitterには検証可能性が備わっていないと言い換えることができる。

Omiの基本的な考え方(Concept) 2コ目

ウェブサービスに関係する登場人物を認証者、アプリ開発者、ホスティングマシーン、干渉者、ユーザーに分ける。それら登場人物間で、互いの利益を最大化する方法を考えていく。

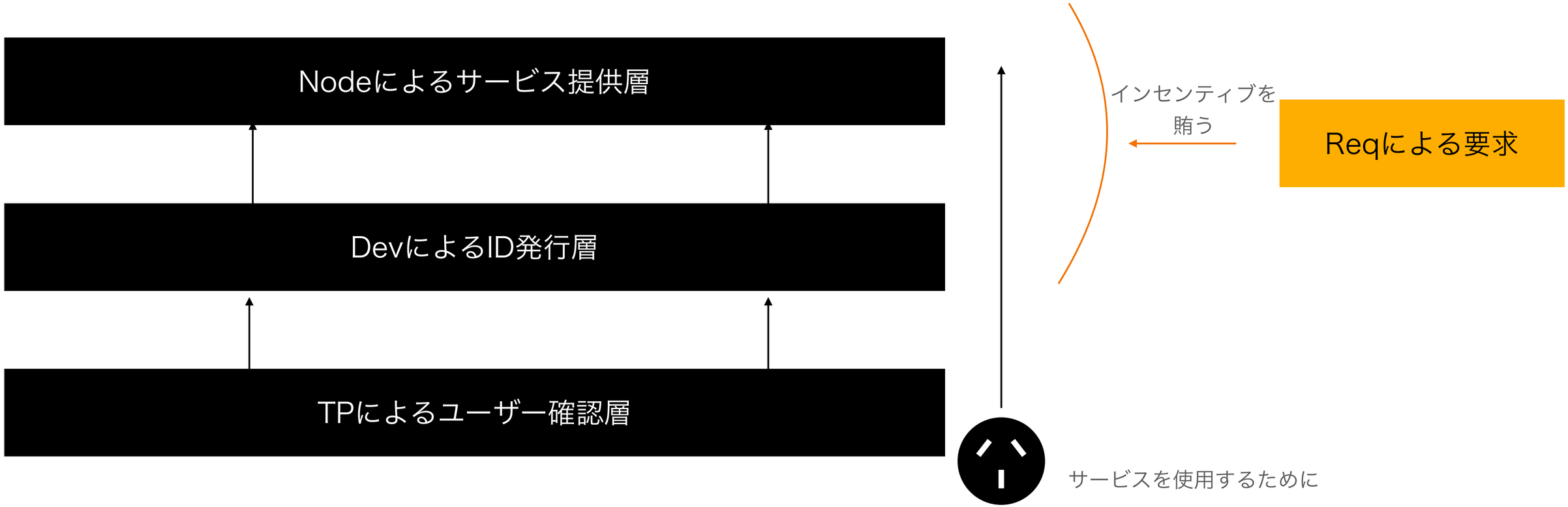
例えばそれぞれの要求が干渉者は「栗きんとんを宣伝したい」ホスティングマシーンは「お金ほしい」ユーザーは「サービス使いたい」とすると、ホスティングマシーンはユーザーに対して干渉者の栗きんとんを宣伝することで、干渉者からお金を受け取る。ユーザーがサービスを使用する限り干渉者は存在する。ホスティングマシーンが干渉者からの利益に納得するとき、システムは持続可能。

登場人物の簡潔な紹介
と、登場人物の言い換え

- 認証者(TP)…ユーザーの電話番号などを確認し、質を担保。
- アプリ開発者(Dev)…UserとNodeのアプリをDSLで開発。お金ほしい。
- ホスティングマシーン(Node)…Devのアプリをホスティング。お金ほしい。
- 干渉者(Req)…NodeとDevに対してリクエストを行う。サービスのユーザーになんらかのリクエストを行いたい。お金はある。

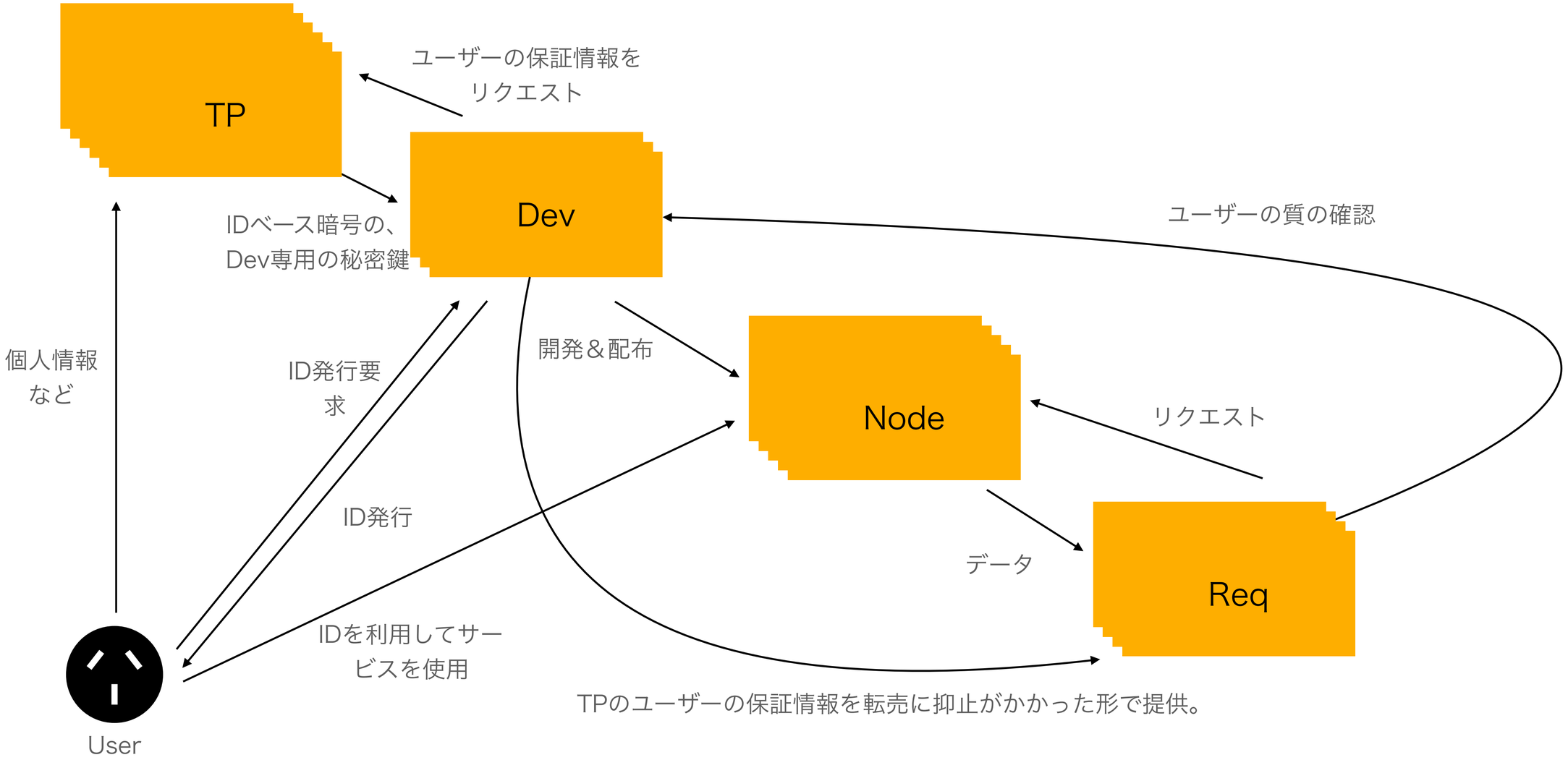
Userは複数のTPにリクエストを行い、１TPあたりに必要な信頼性を低下させる。ユーザーの質を判断する際は、複数のTPからの情報を複合的に判断。

Omiのアーキテクチャの概観 1
レイヤの構築



0miのアーキテクチャの概観 2

このカクカクは主体が複数であるということです。



なぜそんなヘンテコなアーキテクチャか 1
ReqとDevとNodeの関係性

単純なP2Pの場合、DevとNodeのインセンティブは確保されない。そこでネットワークにReqを追加し、DevとNodeのインセンティブ確保を図る。

ReqはNodeになんらかの要求を行い、NodeがUserに要求を伝達。UserはNodeに要求されたデータを挿入。Nodeはそのデータを Reqに売る。ReqはNodeからのデータを元にDevにユーザー情報の照会を行い、信頼性をチェック。Devは対価を受け取る。Nodeから渡されたデータが正常であるとReqが判断した場合、ReqからNodeへの支払いが行われる。

なぜそんなヘンテコなアーキテクチャか 2
ReqとDevの関係性

単純にDevがReqにUserの信頼性情報を販売した場合、ReqがDevの信頼性情報を得た後、その情報を転売する可能性が考えられる。

↓

そのため、DevはTPから受け取った秘密鍵を使用し、Reqの”なんらかの個人情報”を含めた暗号文を作成し、Reqに送信。そうすることで、ReqがUser信頼性の情報を転売することが抑止される。

なぜそんなヘンテコなアーキテクチャか 3
TPとDevの関係性

前提として、UserはDevを信頼しない。というのは、誰でもDevになることができるため、よくわからない誰かに個人情報を渡すことはUserとしてはイヤ（だと思われる）。

だけどもweight assignment problemは考えなければいけない。

↓

Trusted Party(TP)の必要性。DevやReqがTP”群”を信頼することで、Userの信頼性を判断。

なぜそんなヘンテコなアーキテクチャか 4

DevとNodeの関係性

NodeとUserはDevの開発したソフトウェアを掲示板からダウンロードする。ソフトウェアにDevの通信先情報と、公開鍵が埋め込まれている。Userはそれを元にDevにID発行要求を行う。この際、UserはTPの信頼を借りる。UserはNodeにDevの発行したIDを使用して接続。NodeはDevが許可リストにあるかどうかを確認（誰でもDevになることができるため）し、User受け入れの一つの判断材料とする。

Omiのおいしいところ 1

データの内容に対する検証可能性ではない

- ウェブアプリに(誰が発言しているのか検証できるという)検証可能性、検閲耐性、透明性(一部)を持つ。
- ウェブアプリに関わる人物のインセンティブが保たれている状態を持続させることで、ウェブアプリケーションを(使用されている限り)永続化。
- 各登場人物間に必要な信頼性をできる限り低下させる。(信頼性の記述はここでは行わない。最後のリンク参照)

Omiのおいしいところ 2

- Devはソフトウェアを配布し、また誰でもDevになることができるため、ユーザーがサービスを使用する際に特定の主体に依存することがない。(検閲耐性)
- ユーザーが複数のNodeを同様に使用することでアプリに冗長性を追加。
- ユーザーのPrivateなデータはEnd-to-End Encryptionを使い、Publicなデータは署名を使うなどして、ウェブアプリ側が利用可能なデータを明確に区別。(検証可能性 & 透明性) 鍵はユーザー自身の秘密鍵を使用する

Omiのおいしいくないところ

- UserデータをPrivate DataとPublic Dataに大別したが、社会の役に立つ種類の匿名化されたPrivate Data(Big Data)を取得、分析、使用する機構を考えていない。
- **Devの経済的な利益と検閲耐性のメカニズムが対立しちゃった。**誰でもDevになることができるため、誰でも任意のアプリに対してID発行できる。ユーザーが他のDevに対するID発行要求とNodeの他のDev受け入れの判断によって、最初にアプリを開発した者の利益が低下。
- アプリに対する透明性が一部確保できなかった。NodeがUserのPublicなデータをどう使用するかという透明性と、TPとDevが共謀した場合にUserのトラッキングが可能となるというトラッキングに関する不透明性。
- プラットフォームタイプのアプリは大抵カバーできると思うが、購読するタイプのアプリのカバーはReqが不要となったり、UserとNode間の使用料についてなどさらに考えるべきことがある。

最後に

読んでいただき、ありがとうございました！

まだ理論検証の段階です。一部しか書くことができませんでした。
アーキテクチャの詳細は以下で公開しています。

0mi：https://github.com/grvti/0mi_pub/

どんな些細なことでもコメントand/or辛辣な(？)質問お願いいたします。論点がズレているのであれば、各位の貴重なお時間をお取りして申し訳ありませんでした。

Omiがもし有益であれば
ToDo

- 多角的な評価を行なっていない！（考えられる不正の場合について論文に書いたが全場合は列挙不可能。）
  https://blog.ethereum.org/2015/01/28/p-epsilon-attack/ とか、ちゃんと考えるべき、、、

- ユーザーがNodeの情報やDevの情報を得るのは「掲示板」と(誤魔化)した。ウェブアプリの情報のみではなく、DevとNodeの情報を得るための(インセンティブも考慮した)アドレッシングプロトコルが必要。DHTを使用するのが良さげ？ - PUT(<Hex; Dev’s hash>, <String; Node’s IP address>)
  - GET(<Hex; Dev’s hash>) -> List<String; Dev’s IP addresses>, List<String; Node’s IP addresses>