Vol.2022-OS-155 No.11
2022/5/27

明快さと堅牢性を両立するプログラミング言語 definy と GUI 上での Web アプリ開発環境 definy.app

鳴海秀人[†]　岩井将行[†]

†東京電機大学大学院未来科学研究科情報メディア学専攻 〒120–8551 東京都足立区千住旭町 5 番
† E-mail: {narumincho, iwai}@cps.im.dendai.ac.jp

概要：definy は，堅牢な Web アプリケーションをシンプルな仕組みの上で開発することが可能な 2022 年現在開発中の Web アプリケーションと Visual Studio Code の拡張機能だ. definy は，既存の開発環境の問題である，環境構築の煩雑さ，HTML，CSS，JavaScript の仕様に影響を受けた複雑で膨大な言語仕様などの問題を解決する．現在は，Web アプリケーションで入力した型の定義から TypeScript の型定義とバイナリ表現へのコーデックのコード生成と，Visual Studio Code の拡張機能として，definy のコードの編集支援を受けることができる.

キーワード：コード生成，Web，TypeScript，JavaScript，Visual Studio Code Extension

definy.app: A Proposal for a Programming Language combining Clarity and Robustness with a GUI-based Web Application Development Environment

Hideto NARUMI[†] and Masayuki IWAI[†]

† Department of Information Systems and Multimedia Design,
Tokyo Denki University SenjuAsahi-cho 5, Adachi-ku, Tokyo, 120–8551 Japan
† E-mail: {narumincho, iwai}@cps.im.dendai.ac.jp

Abstract: definy is a web application and Visual Studio Code extension under development as of 2022 that enables the development of robust web applications on top of a simple framework. definy solves the problems of existing development environments, such as the complexity of building environments and the large and complex language specifications influenced by the HTML, CSS, and JavaScript specifications. Currently, you can generate code for codecs from type definitions entered in a web application to TypeScript type definitions and binary representations, and as an extension to Visual Studio Code, you can receive editing assistance for your definy code.

Keywords: Code generation, Web，TypeScript，JavaScript，Visual Studio Code Extension

1. はじめに

Web アプリケーションに必要な機能は多い．HTML, JavaScript, 画像などを各ユーザーのブラウザに配信する機能，ユーザーの入力を受け取って画面に情報を表示し，サーバーに HTTP などを介して情報を送信受信する機能，サーバーは受け取ったリクエストからデータを読み込み，データベースなどに保存する機能など多くのことが Web アプリケーションには要求される.

ブラウザで動作するプログラミング言語の JavaScript と WebAssembly 向けのビルドを提供するプログラミング言語は Go, Rust など増えてきたが，画像などのリソース管理，データベースのセットアップなど統一的に管理し開発できる開発環境はあまりない．Web アプリケーションを開発する際に習得しなければならない膨大な量の技術を最小限に抑えるためのプログラミングの抽象化がまだ不足している[1]と言われている．また，フレームワークによってはプログラミング言語以外のテンプレートや設定ファイルが求められ，同じ if 式などの機構にもかかわらず，別の記法を使う必要や，型のチェックが働かず，実行時にエラーが判明することが多い．このような課題を解決するために独自のプログラム言語と開発環境を開発することとなった.

definy は純粋関数型言語として作成している．JavaScript を直接利用するのではなく，関数型言語を利用して web アプリケーションを開発することにはメリットがある [2].

2. Web アプリケーション definy.app

2.1 definy.app の使用方法

URL: https://definy.app/?hl=ja

GitHub リポジトリ: https://github.com/narumincho/definy

旧 definy.app (式の編集と WebAssembly を用いて演算することができる): https://definy-old.web.app/

definy.app は Web ブラウザからアクセスし，definy の機能を利用することができる Web アプリケーションである [図 1]. Web アプリケーションとして作成した理由は，多くの OS と端末で動作させることができるからである．このこの Web ブラウザのみで開発環境が整う機能は，GitHub Codespaces，Gitpod，CodeSandbox，StackBlitz の登場により，より一般的になると考える． Google でソーシャルログインをすることができ，ログインするとプロジェクトを

Ⓒ2022 Information Processing Society of Japan

作成することができる．

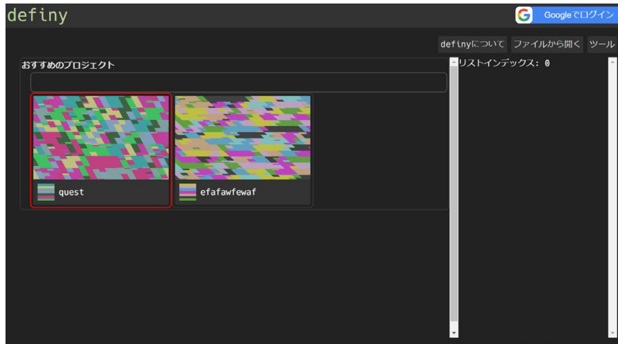

図 1　definy.app のトップページ

プロジェクトを作成した後に，他のプログラミング言語の型定義に対応する型パーツを作成し編集することができる．[図 2]

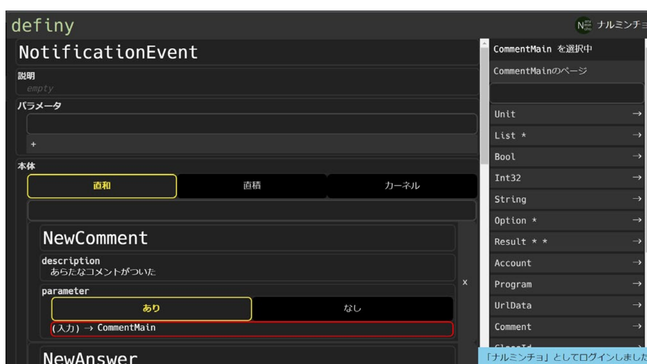

図 2 definy.app の型パーツ編集画面

2.2 システム構成

definy.app のシステム構成は[図 3]のようになっている．機能ごとに複数の GitHub リポジトリに分割していた時期もあったが，現在は narumincho/definy のリポジトリにまとめている．開発言語は TypeScript と少しだけ PureScript で実装している．

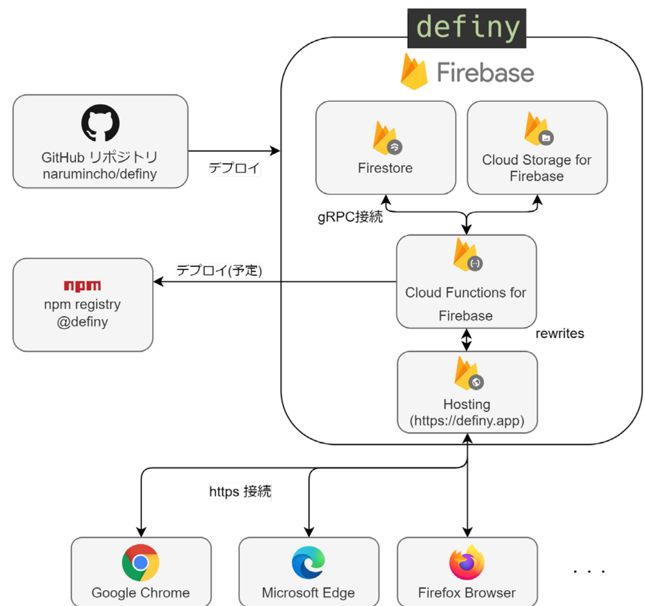

図 3 definy.app のシステム構成

2.3 型パーツの構造

型パーツの編集では，型の名前，説明文，パラメータ（他のプログラミング言語でジェネリクスに相当する），属性，型の本体を編集することができる．

属性（Attribute）は型の定義は，代数的データ型で表現することができるが出力先の言語の組み込みの型で扱ってほしい場合に指定する．現状 AsBoolean と AsUndefined があり，JavaScript と TypeScript に出力した場合は，それぞれ boolean と undefined として扱われる．今後は，Rust や Elm などで登場する，Optional/Maybe，Result をサポートする予定である．

型の本体は，「直和」，「直積」，「カーネル」から選択することができる．Haskell，Rust，Elm などで利用可能な代数的データ型と似ているものとなっている．

直和（Sum）は[図 2]の NotificationEvent のように，「新しいコメントが追加された．コメントの内容は……」か「新しい回答が提出された．回答の内容は……」のどちらかの値をとる場合に使う．パラメータ付き enum（列挙型）とも呼べる．直積で表現するよりも取りうる値の範囲が狭くなり，考慮するパターンが減ることによってバグを生みにくくなる．また一般的なプログラミング言語の代数的データ型では 1 つパターンに複数のパラメータを指定できることが多いが definy では 1 つに制限している．理由は，複数指定する場合は，型の定義を分割したほうが取り扱いやすく，分かりやすいと考えるからである．

直積（Product）は，一般的なプログラミング言語で存在するクラスやレコードに対応するものである．アカウントには「アカウント名」と「アカウント画像」の両方が含まれているのでアカウントを型で表現するなら直積を使うだろう．

カーネル（Kernel）は，直和や直積で表現できないか，

とても非効率な場合に利用するものである．Int32 は 32bit 整数を表現するときに使う．32 ビットで表現できる 4294967296 個のパターンの Sum として表現することもできるが，無駄に定義に容量を使うのでこのような定義の形にした．

作成した型は，プロジェクト詳細ページの「コードを生成する」ボタンから TypeScript の型定義[図 4]と bincode と表現が近い独自のバイナリ形式へのコーデックのコードを生成し利用することができる．[図 5]

```
export type NotificationEvent =
  | { readonly _: "NewComment"; readonly commentMain: CommentMain }
  | { readonly _: "NewAnswer"; readonly answerMain: AnswerMain };
```

図 4 生成されたコード(NotificationEvent の型定義)

```
export const NotificationEvent: {
  /**
   * definy.app 内 の 型パーツの Id
   */
  readonly typePartId: TypePartId;
  /**
   * 独自のバイナリ形式の変換処理ができるコーデック
   */
  readonly codec: Codec<NotificationEvent>;
  /**
   * あらたなコメントがついた
   */
  readonly NewComment: (a: CommentMain) => NotificationEvent;
  /**
   * 新たに回答が提出された
   */
  readonly NewAnswer: (a: AnswerMain) => NotificationEvent;
} = {
  NewComment: (commentMain: CommentMain): NotificationEvent => ({
    _: "NewComment",
    commentMain,
  }),
  NewAnswer: (answerMain: AnswerMain): NotificationEvent => ({
    _: "NewAnswer",
    answerMain,
  }),
  typePartId: "47bbe818062456a8c3aa11c8f59c364c" as TypePartId,
  codec: {
    encode: (value: NotificationEvent): globalThis.ReadonlyArray<number> =>
{
        switch (value._) {
          case "NewComment": {
            return
[0].concat(CommentMain.codec.encode(value.commentMain));
          }
          case "NewAnswer": {
            return [1].concat(AnswerMain.codec.encode(value.answerMain));
          }
        }
      },
      decode: (
        index: number,
        binary: globalThis.Uint8Array
      ): { readonly result: NotificationEvent; readonly nextIndex: number } => {
        const patternIndex: {
          readonly result: number;
          readonly nextIndex: number;
        } = Int32.codec.decode(index, binary);
        if (patternIndex.result === 0) {
          const result: {
            readonly result: CommentMain;
            readonly nextIndex: number;
          } = CommentMain.codec.decode(patternIndex.nextIndex, binary);
          return {
            result: NotificationEvent.NewComment(result.result),
            nextIndex: result.nextIndex,
          };
        }
        if (patternIndex.result === 1) {
          const result: {
            readonly result: AnswerMain;
            readonly nextIndex: number;
          } = AnswerMain.codec.decode(patternIndex.nextIndex, binary);
          return {
            result: NotificationEvent.NewAnswer(result.result),
            nextIndex: result.nextIndex,
          };
        }
        throw new Error("存在しないパターンを指定された 型を更新してく
ださい");
      },
    },
};
```

図 5 生成されたコード
(生成された NotificationEvent のコーデックの部分)

2.4 バイナリ形式の概要とベンチマーク結果

definy.app で生成したコードで扱うバイナリ形式は，直積の場合は順番にエンコードしたもの，直和の場合は，何個目のパターンかの整数とパラメータがあればパラメータをエンコードしたものになる．そのため型が変更された場合に互換性はなくなる．整数のエンコーディングには

LEB128[3]を利用した．

definy.app で生成したバイナリ形式のコーデックで測定したベンチマーク結果がある[4]．結果は次[図 6][図 7]のようになっており，definy のバイナリ形式は，JSON や Protocol Buffers に比べて，サイズが小さいがエンコード，デコードに時間がかるものになっている．
リポジトリ: https://github.com/narumincho/definy-bench

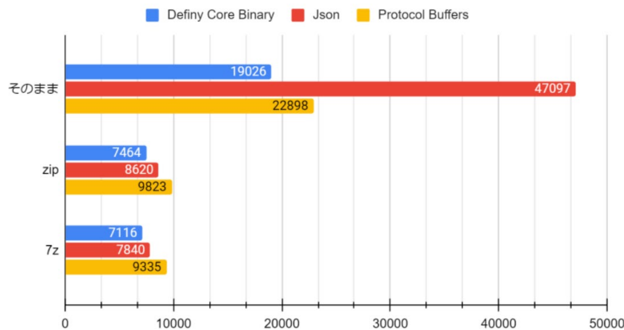

図 6 サイズの比較

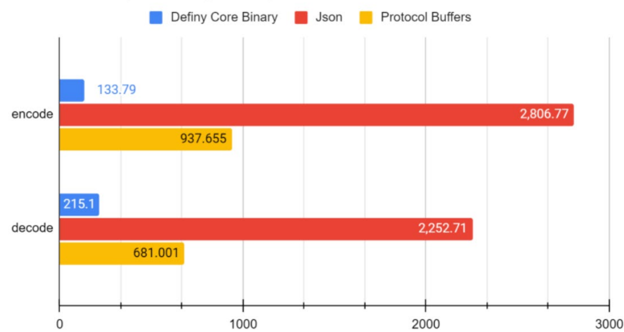

図 7 変換の速さの比較

3. Visual Studio Code 拡張機能版 definy

3.1 使用方法

Visual Studio Code(VSCode)などのエディタの全文検索や置換，補完機能などの編集機能，Git などのバージョン管理システムは開発にとても有益なツールだ．definy は既存の開発環境を置き換えるものとして開発し続けてきたが、definy.app で全文検索やバージョン管理システムの機能を実装し終えるのは難しいと判断し，一旦 Visual Studio Code の拡張機能として開発することに決定した．[図 8]

現在バージョン 0.0.4 として公開している
URL:https://marketplace.visualstudio.com/items?itemName=narumincho.definy

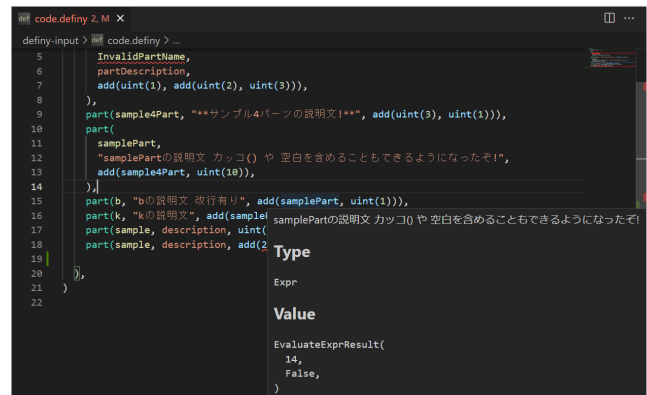

図 8 Visual Studio Code 拡張機能版 definy

Visual Studio Code や vscode.dev などでプライマリーサイドバーから「拡張機能」を選び，definy の拡張機能を検索してインストールボタンを押せばインストールが完了する．VSCode 拡張機能の中の Web Extensions で作成しているので，github.dev や vscode.dev などブラウザ上で動作させることができる[図 9]．Web Extensions として作成する上で，ファイルを読み取るのに require("vscode").fs を使う都合で，直接 VSCode の API を呼んでセマンティックハイライトなどの機能を実装している．そのため Language Server Protocol で実装はしていない．開発言語は PureScript と VSCode API を呼ぶ部分に TypeScript を使用している．

拡張子が .definy のファイルを開くか,「言語モードの選択」から definy を選ぶことによっても，definy として編集が可能となる．

現状可能なことは，コードの意味を理解したうえでのシンタックスハイライト(セマンティックハイライト)，コードフォーマット，補完，ホバーでの情報表示である．式をホバーすればその式の評価結果を確認することができる．

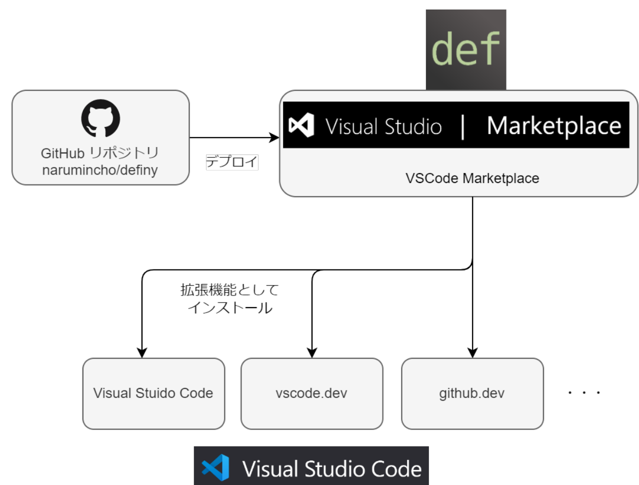

図 9 Visual Studio Code 版 definy のシステム構成図

3.2 言語仕様

構文は

識別子(識別子，識別子，識別子)

の形であり，正規表現 `^[a-zA-Z0-9-]{1,}$` を満たしていない識別子は "(ダブルクォーテーション U+0022) で囲む

必要がある．カッコや空白，カンマは近い見た目文字でも解釈できるようになっている．コードフォーマットで正規化されるので，IME の操作ミスでエラーになることを防げる．

かっこが足りない場合は，補うように解釈される．かっこが過剰な場合は無視する．このようなパーサーであるため，シンタックスエラーは存在しない．

そのためかっこの対応関係がうまく成り立っていないコードをフォーマットするときにエラーで停止することはなく，どんなコードでもコードフォーマットされる．Prettier などの一般的なコードフォーマッターでは構文に問題があるとコードフォーマットされない．また，Prettier と同様にコードの長さによって自動的に改行される．

TypeScript と JavaScript を最初のコンパイルターゲットにしている．コンパイル結果は definy-output フォルダー内に出力される．

現状，数値の足し算とパーツの使用程度しかできないが，今後は文字列操作，画像処理，他の言語の呼び出しなどができるようにし，definy.app の開発言語として置き換える予定だ．

4. 分かりやすくするためのアイデア

definy.app でこれから実装する機能のアイデアがいくつかある．

definy.app では画像や動画，音声などのデータを，ファイルシステムを介さずパーツ(定数)として定義することができ，正確な表現が可能となる．現状の開発環境やプログラミング言語では画像などのテキストではないリソースを扱うときは，ファイルシステムを介す必要があり，ファイルシステムを扱う場合には実行前にファイルが存在するか，中身が条件を満たしていて正常なものかのチェックがあまり行われていない．

definy.app ではコードは文字列ではないため，キーボードから入力された文字を加工して表示することが可能である．例えば，IEEE754 の倍精度の浮動小数点数で 0.1 を正確に表現することは仕様上できず，0.1+0.2 が 0.3 と等しくないという現象が発生するが，definy では，0.1 と入力したものは，右側の detail view で「0.1000000000000000055511151231257827021181583404541015625」という正確な表現を確認することができ，0.1 が正確に 0.1 ではないということに気づきやすくなっている．

ユーザーが作るリテラルエディタには独自のバリデーションの処理も可能なため，クリア可能な迷路のデータなど，型レベル演算のような型の指定だけでは防ぐことが困難な入力とそのバリエーションをエディタ上でさせることもできる．

現状 Rust でコードを以下のように unwrap をする場面がよく登場する．

```
use regex::Regex;
let re = Regex::new(r"^¥d{4}-¥d{2}-¥d{2}$").unwrap();
assert!(re.is_match("2014-01-01"));
```

https://docs.rs/regex/latest/regex/ のサンプルコード

これは，Regex::new に与えられる文字列が正当な正規表現とは限らないため，Result<Regex, Error> 型が返ってきており，それを unwrap しているのである．不正な正規表現(例えば [ )を指定した場合は，実行時エラーとなりプログラムは停止する．

JavaScript のように正規表現リテラルを用意すれば良いという話ではあるが，開発するアプリごとにカスタマイズした検査機能が必要になった場合に機能を追加することができない．definy のカスタムリテラルを利用することによって正規表現のエラーをコード記入時に表示でき，コードフォーマットをすることができる．Rust のような unwrap による実行時エラーを気にしなくてよくなる．

Rust では，マクロを利用すればコンパイル時に正規表現のエラーを検知することは可能ではあるが，実装難易度が高いためか，多くのライブラリでは実装されていない．

パーツの使用などは選択肢から選ぶので，間違って存在しないパーツの参照を入力することは減る．

GUI 上で選択したものは文字列として扱うのではなく構造化されたデータとして扱うことができる．型が List Int のデータをコピー&ペーストする場合に，文字列のコピーでは区切り文字や数値の文字列表現を気にする必要があるが，definy の場合はそのままの値でコピー&ペーストすることができるので，正確な値のまま操作することができる．

そのほかの仕様は Elm を参考にしている．Elm は関数型リアクティブプログラミング言語として登場し UI を記述する上で大きなメリットがあると言われていた[5]が，現在の Elm は The Elm Architecture というシンプルな関数を組み合わせてアプリケーションを表現するものとなっており[6]，definy はこの新しい Elm を参考にしている．オブジェクトなどの参照による破壊的変更がないイミュータブルなデータ構造，代数的データ型による現状の正確な表現，関数は入力が同じなら，どの環境でも必ず同じ結果を返す純粋な関数の特徴を definy は受け継ぐ．

5. まとめと今後の展望

VSCode 版 definy の開発を進め，数値の計算だけでなく画像を出力できるようにし，同じ画像を出力するプログラムを Python や TypeScript などの普及しているプログラミング言語と definy で実装してもらう実験をする予定である．

最終的には definy.app でも Git 以上に使いやすいバージョン管理システムやコミュニケーションツールを開発し内蔵させて，VSCode 拡張機能版 definy は廃止する予定だ．

6. 参考文献

[1] Manuel Serrano, Erick Gallesio, Florian Loitsch,“Hop, a Language for Programming the Web 2.0” OOPSLA’06 October 22–26, 2006, USA.

[2] Yoo, Daniel, ”Building Web Based Programming Environments for Functional Programming” : Doctoral dissertation, Worcester Polytechnic Institute, Worcester, Massachusetts, USA: : 2012

[3] “LEB128 – Wikipedia“ https://en.wikipedia.org/wiki/LEB128 2022 年 5 月 06 日

[4] 鳴海秀人, “代数的データ型から型定義とバイナリ形式へのデコーダとエンコーダのコード生成” ,情報処理学会論文誌プログラミング(PRO),14(2),28-28,2021.

[5] Evan Czaplicki : ” Elm: Concurrent FRP for Functional GUIs ” : Senior thesis Harvard University, Cambridge, USA : 30 : 2012

[6] Evan Czaplicki , “Elm - delightful language for reliable web applications”, https://elm-lang.org/ 2022 年 5 月 02 日参照