VDM-SL 実行可能仕様による Web API プロトタイピング環境

小田 朋宏
株式会社 SRA
tomohiro @ sra.co.jp

荒木 啓二郎
九州大学
araki @ csce.kyushu-u.ac.jp

要旨

ソフトウェアシステムの仕様を厳密に記述することで仕様記述の問題点をより早期に発見し修正する開発手法として，形式手法が注目されている．本稿では，形式的仕様記述言語 VDM-SL の実行系を使って実行可能仕様を WebAPI として提供するウェブサーバ Webly Walk-Through を紹介する．

1. はじめに

軽量形式手法[1][2]はソフトウェアを経済的に開発するための形式手法の実践として注目されている．VDM-SL[3][4]は軽量形式手法に適した仕様記述言語として多くの成功事例が報告されている．それらの成功事例では，VDM-SL で記述された仕様が多くの工程の技術者によって信頼できる文書として利用されたことが報告されている[5][6].

軽量形式手法では，形式手法が持つ厳密さおよびツール支援を，開発工程全体のうちより効果的な部分に適用する．自然言語が持つ曖昧さを軽減できることが挙げられる．構文検査や型検査，証明課題の自動生成によるツール支援も大きな利点として支持されている．VDM-SL は実行可能なサブセットを持ち，実行系を利用してインタプリタ実行することができる.

複数の VDM の成功事例で実行系が有効に用いられている．仕様をインタプリタ実行することは，要求獲得やテストやマニュアル作成などに効果的に利用されている[6][7].

我々は VDM-SL 実行系を利用したツールを開発してきた．Smalltalk[8]上の外部インタプリタ接続ライブラリ SOMETHINGit[9][10]を使ったライブ UI プロトタイピング環境を構築し，また，仕様記述の初期段階での試行錯誤を支援するウェブベース仕様記述環境 VDMPad[11][12][13]を開発した.

我々はソフトウェア開発全体における実行系の利用シーンを増やすことにより，形式的仕様記述をより効果的に導入することができると考えている．本稿では，VDM-SL による実行可能仕様の新たな利用シーンとしてウェブアプリケーションに注目し，ウェブアプリケーションの開発において重要な Web API のプロトタイピング環境として開発した Webly Walk-Through を紹介する.

2. Web API 仕様記述言語としての VDM-SL

VDM-SL はモデル規範型の形式的仕様記述言語で，開発対象となるシステムを型，定数値，関数，状態，操作を定義することで記述する．ここでは VDM-SL の言語機能の概要を紹介し，Web API の仕様記述の具体例を紹介する.

2.1. VDM-SL の概要

VDM-SL ではシステムをモジュールに分割し，各モジュールに型，定数値，関数，状態変数，操作を定義する．ここで，関数は参照透明な関数で，操作は状態変数への参照および状態変数への代入を伴う手続きである．関数および操作は，事前条件と事後条件のみを定義した陰仕様と，事前条件および事後条件に加えて処理本体を明示的に定義した陽仕様のいずれかで定義する．陽仕様で定義された関数および操作は実行系によりアニメーション実行することができる．各モジュールにおいて imports 宣言および exports 宣言によりモジュールのインターフェイスを定義し，モジュールをまたがる参照関係を記述する．インターフェイスでは型，定数値，関数，操作を公開することができるが，状態変数を公開することはできない.

VDM-SL 仕様の例として，メールアドレス管理システムの実行可能仕様を示す．メールアドレス管理システムは個人の名前とメールアドレスを管理する．また，各個人はグループを作ることができる．グループはグループ名とメンバーからなるものとする．1 人の個人は任意の数のグループに所属することができるものとする．ユーザは自分と同じグループに属するメンバーの名前とメールアドレスに

```
module AddressBook
exports
  types ID; NAME; EMAIL;
  operations add:NAME*EMAIL==>(); id:NAME==>[ID];
             name:ID==>[NAME]; email:ID==>[EMAIL];
             names:()==>set of NAME;
definitions
types
  ID = nat;
  NAME = seq of char;
  EMAIL = seq of char;
  ITEM :: name:NAME email:- EMAIL
state AddressBookState of
  book : inmap ID to ITEM
  next_id : nat
  init s == s = mk_AddressBookState({|->}, 0)
end
operations
  add : NAME * EMAIL ==> ()
  add(name, email) == let item = mk_ITEM(name, email) in
    book(if item in set rng book
         then (inverse book)(item)
         else get_next_id()) := item;
  name : ID ==> [NAME]
  name(id) ==
    return if id in set dom book then book(id).name else nil;
  email : ID ==> [EMAIL]
  email(id) ==
    return if id in set dom book then book(id).email else nil;
  id : NAME ==> [ID]
  id(name) == return let item = mk_ITEM(name, "") in
    if item in set rng book then (inverse book)(item) else nil;
  names : () ==> set of NAME
  names() == return {name | mk_ITEM(name, -) in set rng book};
  get_next_id : () ==> nat
  get_next_id() == let id = next_id in (next_id := id + 1; return id);
end AddressBook
module Groups
imports from AddressBook types ID
exports all
definitions
types
  NAME = seq of char
state GroupsState of
  groups : map NAME to set of AddressBook`ID
  init s == s = mk_GroupsState({|->})
end
operations
  add : NAME * AddressBook`ID ==> ()
  add(name, id) == groups(name) :=
    (if name in set dom groups then groups(name) else {})
    union {id};
  remove : NAME * AddressBook`ID ==> ()
  remove(name, id) ==
    if name in set dom groups
    then (groups(name) := groups(name) \ {id};
          if groups(name) = {} then groups := {name} <-: groups);
  member : NAME * AddressBook`ID ==> bool
  member(name, id) ==
    return name in set dom groups and id in set groups(name);
  friends : AddressBook`ID ==> set of AddressBook`ID
  friends(id) ==
    return dunion {s | s in set rng groups & id in set s} \ {id};
end Groups
```

リスト 1:メールアドレス管理システムの VDM-SL 仕様

アクセスすることができるものとする. リスト 1 に VDM-SL による仕様記述の例を示す.

上記のメールアドレス管理システムは 2 つのモジュールから成る. AddressBook モジュールは, 個人に ID を割り当てて名前とメールアドレスを管理する. 個人を登録する add 操作および ID から名前をひく name 操作, ID からメールアドレスをひく address 操作を提供する.

Groups モジュールはグループに関する管理をおこなう. 個人をグループに登録する add 操作, 個人の登録をグループから取り消す remove 操作, 個人がグループに属するかを検査する member 操作, 指定された個人と 1 つ以上の共通グループに属する個人の集合を求める friends 操作が提供される.

上記の VDM-SL 仕様は実行系によってインタプリタ実行することができる. 実行を開始すると, 個人およびグループがそれぞれ空の状態でスタートする. 開発者はインタプリタ上で

AddressBook`add("Adam", "adam@example.com")

を実行することで Adam のメールアドレス adam@example.com を登録するなどして記述した仕様が求められる機能を提供しているか妥当性の確認をすることができる.

2.2. Web API 記述の例

Web API の仕様では, 各 API の入力および出力のデータ型, 入力への制約条件, 処理内容が明示されることが重要である.

前述のシステムの VDM-SL 仕様はレポジトリとしての機能を記述したものである. 次にウェブブラウザをクライアントと仮定して, HTTP リクエスト/レスポンスを通してレポジトリの機能をクライアントに提供する Web API の仕様を VDM-SL で記述する. 前節で定義したレポジトリの VDM-SL 仕様から必要な機能を参照することで, 実行可能な Web API 仕様を記述することができる.

リスト 1 の仕様に追加する Web API の仕様をリスト 2 に示す. リスト 2 の API モジュールは, API で利用するデータ型 ENTRY および MEMBER を定義し, それらデータ型を入力および出力とする操作として register, join, leave, list の 4 つの公開操作を定義する. 各 API 仕様は入出力のデータ型および入力値への制約(pre に続いて記述された事前条件)が明示され, 条件を満たした入力が AddressBook モジュールおよび Groups モジュールが提供するどの操作に与えられるかが明示される. API モジュールが定義する 2 つのデータ型は JSON 形式に対応

```
module API
imports
  from AddressBook operations add; id; name; email; names,
  from Groups operations add; remove; member; friends
exports
  types ENTRY; MEMBER;
  operations register:ENTRY==>(); update:ENTRY==>();
             join:MEMBER==>(); leave:MEMBER==>();
             list:seq of char==>seq of ENTRY;
definitions
types ENTRY :: name : seq of char email : seq of char;
types MEMBER :: name : seq of char group : seq of char;
operations
register : ENTRY ==> ()
register(mk_ENTRY(name, email)) ==
  AddressBook`add(name, email)
  pre name not in set AddressBook`names();
update : ENTRY ==> ()
update (mk_ENTRY(name, email)) ==
  AddressBook`add(name, email)
  pre name in set AddressBook`names();
join : MEMBER ==> ()
join(mk_MEMBER(name, group)) ==
  Groups`add(group, AddressBook`id(name))
  pre name in set AddressBook`names();
leave : MEMBER ==> ()
leave(mk_MEMBER(name, group)) ==
  Groups`remove(group, AddressBook`id(name))
  pre name in set AddressBook`names();
list : seq of char ==> seq of ENTRY
list(name) == return
  let s:set of nat = Groups`friends(AddressBook`id(name)) in
    [mk_ENTRY(AddressBook`name(i), AddressBook`email(i))
    | i in set s]
  pre name in set AddressBook`names();
end API
```

リスト 2:VDM-SL による Web API 仕様

させることができる. 例えば ENTRY 型は name および email の 2 つの文字列のフィールドを持つレコード型として定義されているが, JSON フォーマットでは name および email の 2 つのタグを持つオブジェクトデータに対応させることができる.

VDM-SL の陽仕様で記述された Web API の仕様はインタプリタ実行が可能であり, 具体的な入力データを与えて実行することにより妥当性を確認することができる.

3. Webly Walk-Through

インタプリタ実行により妥当性が確認された Web API 仕様は, Web API サーバを実装する開発者と Web API のクライアントを実装する開発者により Web API 仕様書として参照される. Web API サーバのホワイトボックステストにおいて Web API 仕様書がテスト技術者に参照されるだけでなく, Web API 仕様書が実行可能であることから, Web API 仕様の実行結果を Web API サーバの実装のテストに利用することができる.

Webly Walk-Through は, HTTP サーバと VDM-SL 実行系を組み合わせ, HTTP リクエストから VDM-SL の評価式に変換し, 評価結果から HTTP レスポンスを生成しクライアントに応答することで, JavaScript 等で記述されたクライアントプログラムから呼び出すことを可能にする Web API プロトタイピングシステムである. Webly Walk-Through によって WebAPI 仕様をクライアントプログラムから利用可能にすることで, クライアント技術者は Web API の実装完了を待たずに並行してクライアントプログラムの開発に Web API を利用することができる.

3.1. アーキテクチャ

図 1 にシステム構成を示す. Webly Walk-Through は MacOSX および Linux 上で動作する HTTP サーバである. 実装言語は Squeak Smalltalk[8]であり, VDM-SL 実行系として VDMJ[14]を利用する. Squeak Smalltalk と VDMJ のブリッジには, 多言語 UI プロトタイピングライブラリ SOMEHTINGit[9][10]を利用した.

3.2. HTTP サーバおよびユーザインターフェイス

Webly Walk-Through は HTTP サーバとして以下の 3 種類のサービスを提供する.

- ウェブベース開発環境
- 静的ファイルサービス
- Web API サービス

Webly Walk-Through は VDM-SL 仕様で記述された Web API をインタプリタ実行するだけでなく, それ自体がウェブベースの VDM-SL 開発環境およびウェブコンテンツ編集環境を提供する. Webly Walk-Through のウェブベース開発環境は以下の 3 つの画面を持つ.

- VDM-SL 仕様および VDM-SL/JSON 相互変換規則を記述し, 状態変数を監視し, VDM-SL 評価式を実行する VDM-SL 開発画面(図 2)

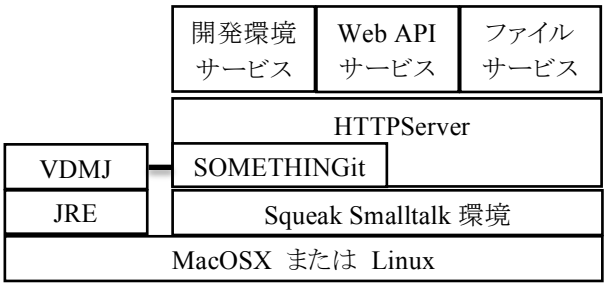

図 1: Webly Walk-Through のシステム構成

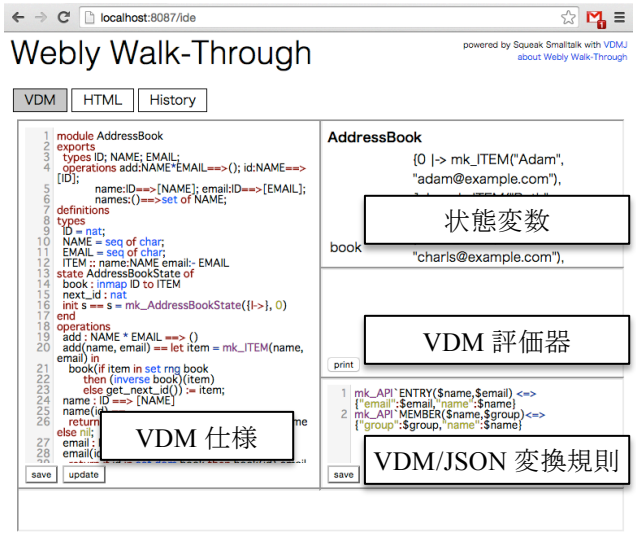

図 2: VDM-SL 開発画面

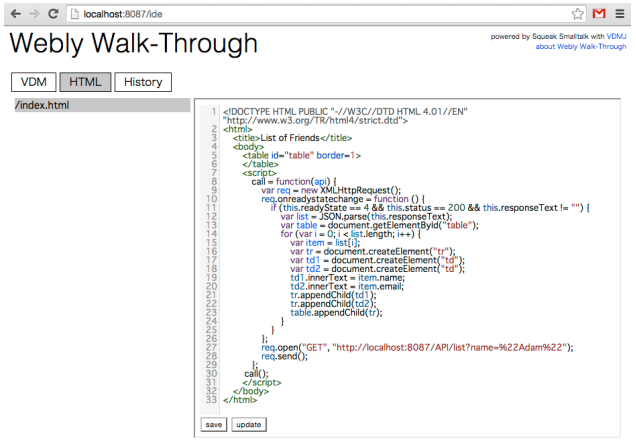

図 3:ファイル編集画面

図 4: 実行履歴画面

- ◆ 静的ファイルを編集するファイル編集画面 (図 3)
- ◆ Web API 仕様の実行履歴を表示する履歴画面 (図 4)

図 2 に示した VDM-SL 開発画面の VDM-SL 仕様部分では，Web API を提供している VDM-SL ソースを編集することができる．ただし，Webly Walk-Through 上での編集は細かな修正を想定しており，第 2 章で紹介したような VDM-SL 仕様記述をスクラッチから記述するのは VDMTools[15][16]や Overture tool 等の VDM 統合環境上で行うことを想定している．

VDM-SL 開発画面では，インタプリタ実行中の状態変数を監視することができる．各モジュールの各状態変数の現在の値が表示される．VDM-SL 評価機部分では，VDM-SL の表現式を評価することで，Web API のクライアントプログラムについて特定の状態をサーバ側に作り出してテストすることができる．

Web API は VDM-SL で記述されているが，クライアントプログラムで Web API に入出力するデータフォーマットとして JSON を利用することができる．VDM-SL/JSON 変換規則を簡単なパターン言語で記述することができる．パターン言語の詳細は次節で説明する．

3.3. Web API の処理

Webly Walk-Through は与えられた VDM-SL 仕様の全モジュールの全操作および全関数を URL の path 部

/ <モジュール名> / <操作・関数名>

で公開する．操作や関数への引数は URL の query 部もしくは POST メソッドのボディ部にフォーム送信フォーマット (application/x-www-form-urlencoded) で送られる．各引数値は JSON フォーマットで表現される．

Webly Walk-Through は Web API クライアントから送信された HTTP リクエストからモジュール名，操作名，引数を取得し，各引数を JSON フォーマットから VDM-SL の表現式に変換し，下記の形式の VDM-SL 評価式を合成する．

<モジュール名>`<操作・関数名>(<引数 1>,…,<引数 n>)

VDM-SL 実行系 VDMJ が上記評価式を実行し，返り値を JSON フォーマットに変換し，HTTP レスポンスのボディ部に格納し，Web API クライアントに送信する．

リスト 2 で定義された API モジュールの list 操作を Web API を通して実行する例を図 5 に示す．

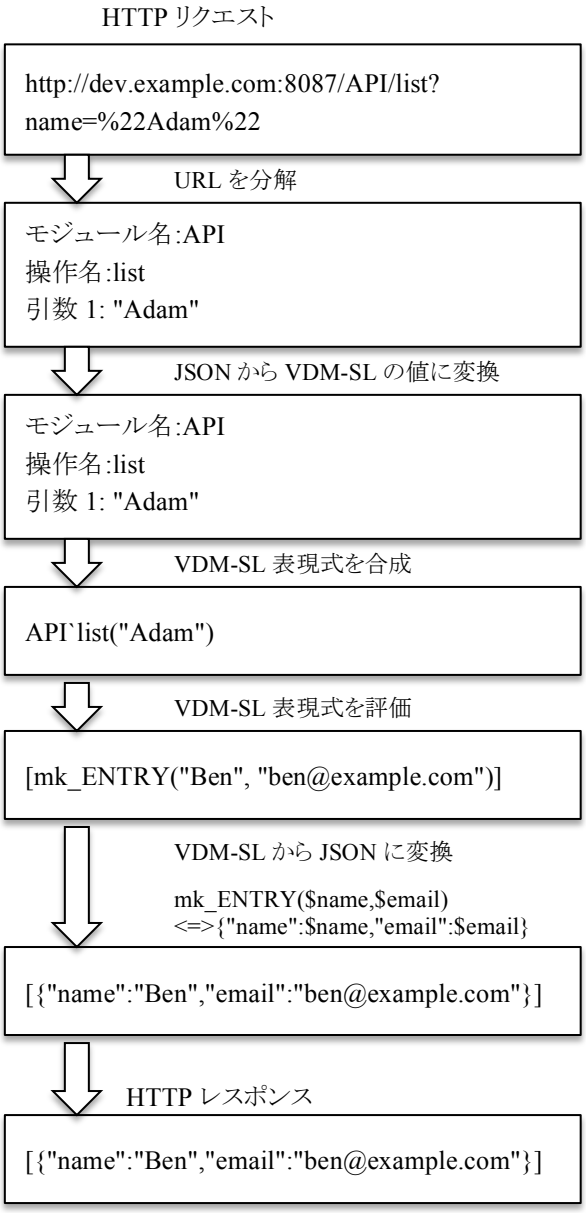

図 5: Web API **の処理の流れ**

4. まとめ

軽量形式手法は高信頼性を求められるソフトウェア開発のみでなく，経済性を求められる開発にも適用されてきた．VDM-SL は仕様に関する検証だけでなく，実行系によるインタプリタ実行が可能であり，仕様書として読むだけでなく，テストオラクルやプロトタイプとして利用することも可能にする．Webly Walk-Through は VDM-SL 仕様の応用をウェブアプリケーション開発における Web API を VDM-SL で記述し，それを Web API としてクライアントプログラムから利用可能にするものである．

ウェブアプリケーション以外にも VDM-SL による実行可能仕様を応用できる領域は多く存在している．Webly Walk-Through は実開発プロジェクトには使用されていない．今後は Webly Walk-Through をはじめとする実行可能仕様を応用したツールの実プロジェクトへの適用を図るとともに，さらなる新しい応用領域のためのツールの開発を継続する．

5. 謝辞

本研究を遂行するにあたって中小路久美代氏，山本恭裕氏，大森洋一氏，日下部茂 氏，佐原伸氏，酒匂寛氏，栗田太郎氏より多くの助言を受けた．ここに謝意を記す．本研究の一部は，JSPS 科研費(24220001) の助成を受けたものである．

参考文献

[1] J. Fitzgerald, P. G. Larsen, and S. Sahara, “VDMTools: Advances in Support for Formal Modeling in VDM,” *ACM Sigplan Notices*, vol. 43, no. 2, pp. 3–11, February 2008.

[2] S. Agerholm and P. G. Larsen, “A Lightweight Approach to Formal Methods,” in *Proceedings of the International Workshop on Current Trends in Applied Formal Methods*. Boppard, Germany: Springer-Verlag, October 1998.

[3] J. Fitzgerald and P. G. Larsen, *Modelling Systems - Practical Tools and Techniques in Software Development*. The Edinburgh Building, Cambridge CB2 2RU, UK: Cambridge University Press, 1998.

[4] P. G. Larsen, B. S. Hansen et al., "Information technology - Programming languages, their environments and system software interfaces - Vienna Development Method - Specification Langauge - Part 1: Base language," December 1996, International Standard ISO/IEC 13817-1.

[5] J. Woodcock, P.G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal Methods: Practice and Experience," ACM Computing Surveys, vol. 41, no. 4, pp. 1-36, October 2009

［6］ IPA/SEC, 厳密な仕様記述における形式手法成功事例調査報告書, 独立行政法人情報処理推進機構 技術本部 ソフトウェア・エンジニアリング・センター, 2013
http://www.ipa.go.jp/sec/softwareengineering/reports/20130125.html

［7］ T. Kurita and Y. Nakatsugawa, "The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip," Intl. Journal of Software and Informatics, vol. 3, no. 2-3, pp. 343-355, October 2009.

［8］ D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay, “Back to the future - the story of squeak, a practical smalltalk written in itself,” *ACM SIGPLAN Notices*, vol. 32, no. 10, pp. 318–326, 1997.

［9］ T. Oda, K. Nakakoji, and Y. Yamamoto, “SOMETHINGit: A Prototyping Library for Live and Sound Improvisation,” in *Proceedings of the 1st International Workshop on Live Programming*, ser. LIVE ’13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 11–14.

［10］小田朋宏, 中小路久美代, 山本恭裕, ライブ UI プロトタイピングに向けたマルチ言語環境 SOMETHINGit, ソフトウェアシンポジウム2013論文集, 2013

［11］T. Oda and K. Araki, “Overview of VDMPad: An Interactive Tool for Formal Specification with VDM,” in *International Conference on Advanced Software Engineering and Information Systems (ICASEIS) 2013*, Nov 2013.

［12］小田朋宏, 荒木啓二郎, 形式的仕様スケッチのためのウェブベース開発環境 VDMPad, ソフトウェアシンポジウム 2014 論文集, pp.139-146, 秋田, Jun 9-11, 2014

［13］T. Oda, K. Araki, P. G. Larsen, "VDMPad: a lightweight IDE for Exploratory VDM-SL Specification", *in FME Workshop on Formal Methods in Software Engineering (FormaliSE) 2015 (to appear)*

［14］N. Battle, “VDMJ User Guide,” Fujitsu Services Ltd., UK, Tech. Rep., 2009.

［15］P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl, and M. Verhoef, “The Overture Initiative – Integrating Tools for VDM,” *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 1, pp. 1–6, January 2010.

［16］K. Lausdahl, P. G. Larsen, and N. Battle, “A Deterministic Interpreter Simulating A Distributed real time system using VDM,” in *Proceedings of the 13th international conference on Formal methods and software engineering*, ser. Lecture Notes in Computer Science, S. Qin and Z. Qiu, Eds., vol. 6991. Berlin, Heidelberg: Springer-Verlag, pp. 179–194, October 2011