VDM-SLによるソフトウェアアーキテクチャの記述法

張 漢明　野呂 昌満

南山大学 数理情報学部

本論文では、ソフトウェアアーキテクチャにおけるソフトウェアコンポーネント間の関係を分析し、ソフトウェアアーキテクチャの形式的な仕様記述の枠組みについて議論する。住所録システムの事例を通して、同一レベルのソフトウェアの構成要素間の「関係」と、異なった抽象レベルのソフトウェアの構造間の「関係」を明らかにして、形式仕様記述言語 VDM-SL による記述例を示す。

1　はじめに

オブジェクト指向技術に基づいたソフトウェアアーキテクチャによるソフトウェア開発は、ソフトウェア開発のコストを少なくするための有効な手段の一つである。ソフトウェアアーキテクチャは、要求、設計、および実現等の異なった抽象レベルにおけるソフトウェアの構造を表現する。しかしながら、ソフトウェアを構成する構成要素間の「関係」や、抽象レベルの違う構造の間の「関係」を表現するための有効な表記法はない。

本研究の目的は、ソフトウェアアーキテクチャにおける、同一レベルのソフトウェアの構成要素間の「関係」と、異なった抽象レベルのソフトウェアの構造間の「関係」を明らかにして、その記述法を提示することである。ソフトウェアアーキテクチャにおいて抽象度に関する定義は明確ではない。抽象度の基準を明確にするためには、形式的な記述に基づいた議論が不可欠である。

本研究の基本アイデアは、オブジェクト指向に基づいたコンポーネントの仕様を形式的に表現し、コンポーネント間の関係を形式的な仕様記述の間で関連付けることである。ソフトウェアアーキテクチャに形式手法を適用することにより、より厳密なアーキテクチャの記述法を提示することを目指す。形式仕様記述言語は、実際の開発現場で利用されることを想定して、最も広く利用されている言語の一つである VDM-SL を導入した。

本論文では、住所録システムを事例として、VDM-SL による要求レベルと設計レベルのコンポーネントの仕様を記述し、それぞれのコンポーネント間の関係について議論する。構成要素におけるデータの型に着目して、コンポーネント間の関係を論じる。抽象度の違いは、データの型を基準として明確にすることができる。

2　ソフトウェアアーキテクチャ

ソフトウェアアーキテクチャ[1] は、ソフトウェアの構造を示すだけではなく、ソフトウェア開発のプロセスを内包するものである。ソフトウェアアーキテクチャの構成要素の抽象度に関する定義は曖昧であり、一般的には、同一抽象度の構成要素を持つと理解されている。しかし、図 1 に示すように、ソフトウェアアーキテクチャは、異なる抽象度の構成要素を集めたものであると考える。

ソフトウェアを開発する際には、様々な抽象レベルで対象システムを捉えて分析する。実際のソフトウェア開発では、(1) 要求仕様、(2) 設計仕様、(3) 実現 (プログラム) 仕様の異なった抽象レベルの記述がある。

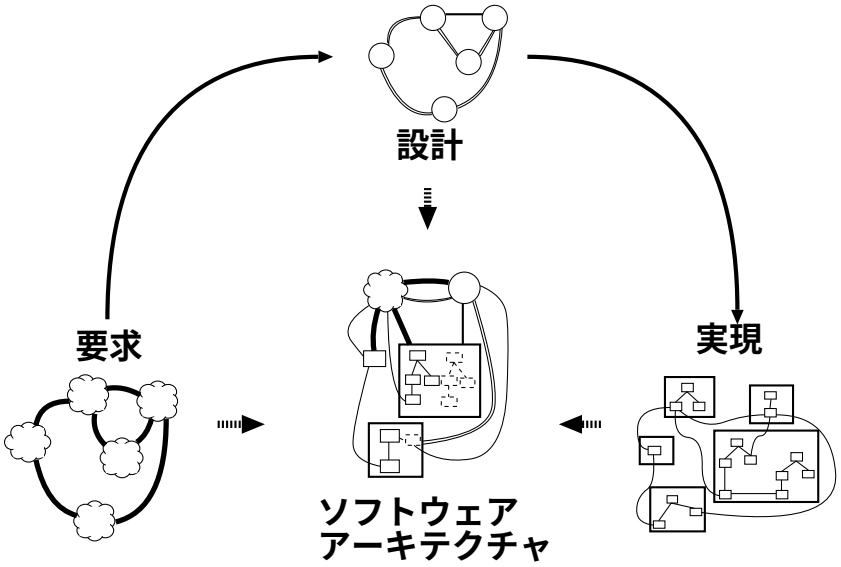

図 1: ソフトウェアアーキテクチャの概念

それぞれの抽象レベルの記述の中では、更に異なった抽象レベルの記述があるかも知れない。しかし、抽象レベルの基準を明確に規定して、洗練された記述を構築することは易しい作業ではない。例えば、要求仕様において、記述対象の抽象度のレベルに対する共通認識がなく、要求仕様において抽象的な記述と詳細な記述が混在し、要求の本質を理解する妨げとなっている場合が多く見受けられる。

ソフトウェアアーキテクチャは、様々な抽象レベルにおいて、ソフトウェアのコンポーネントとコンポーネント間の関係を記述する。したがって、コンポーネント間の関係には、

(1) 異なる抽象レベルのコンポーネント間の関係と

(2) 同じ抽象レベルのコンポーネント間の関係

が存在する。抽象度の基準とコンポーネント間の関係を明示することができれば、要求、設計、および実現のそれぞれの開発工程における作業内容が明確になる。ソフトウェアアーキテクチャは、異なる抽象度の構成要素を集めたものであるから、ソフトウェアアーキテクチャが決まれば、ソフトウェアの開発プロセスを明示したことになる [3]。抽象度の基準を明確にするためには、形式手法の導入が有効な手段である。

3 形式手法適用のアイデア

形式手法とは、コンピュータシステムのモデル化、設計、解析を行うための数学を基にした技術であり、形式手法の有用性については、文献 [4] において明解に述べられている。仕様を形式的に記述することの利点の一つは、厳密さと抽象化により仕様記述の基準を明示することである。オブジェクト指向コンポーネントの仕様を形式的に表現して、仕様の上でコンポーネント間の関係を示すことにより、コンポーネント間の関係を議論する。

本研究における形式手法の適用の概念を図 2 に示す。コンポーネント間の関係は、形式仕様コンポーネント記述の間で明示する。抽象度 A と抽象度 B には、それぞれの抽象度においてコンポーネント間の関係が存在する。また、抽象度 A と抽象度 B の異なる抽象度において、コンポーネント間の関係が存在する。さらに、形式仕様コンポーネント記述とオブジェクト指向コンポーネント記述の間にも、何らかの関係があると考えられる。次章では、具体的な事例を用いて、これらの関係について議論する。

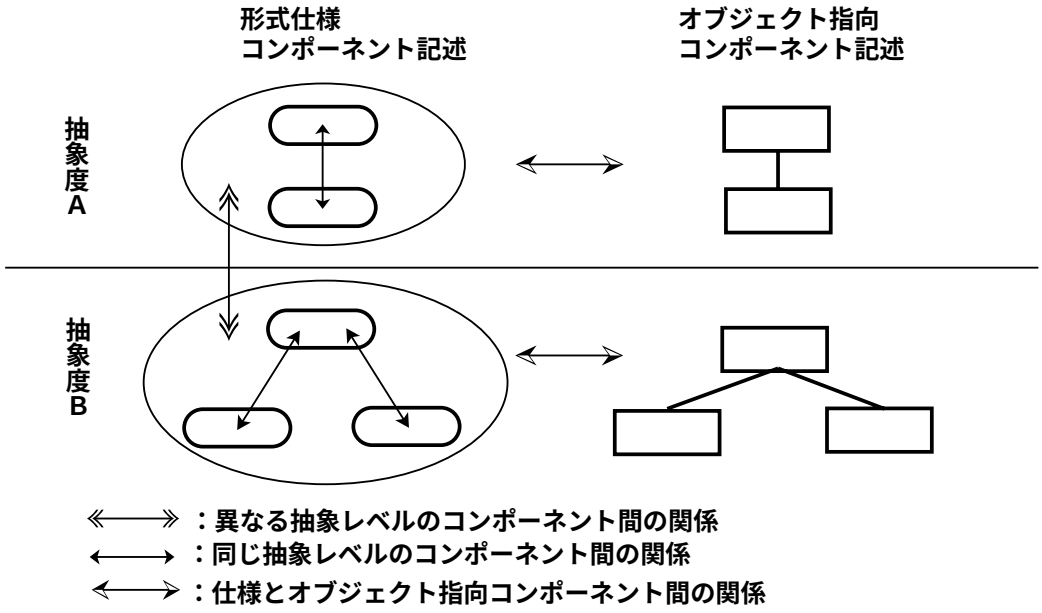

図 2: 形式手法適用の概念

4 住所録システムの事例

本章では、住所録システムを事例として、VDM-SL によるコンポーネントの形式的な仕様記述の例を示し、コンポーネント間の関連について考察する。住所録システムとは、名前と住所の対応を記録するシステムである。ここでは機能として、住所の新規登録と、名前による住所の検索の二つの機能について考える。また、仕様記述のレベルとして、(1) アプリケーションの抽象仕様記述、(2) 要求レベルのコンポーネントの仕様記述、および (3) 設計レベルのコンポーネントの仕様記述の三つの段階の仕様記述を試みる。VDM-SL の表記は、文献 [2] で用いられている表記法に準拠する。

4.1 アプリケーションの抽象仕様記述

アプリケーションの抽象仕様記述では、ソフトウェアの構造や実現のことは考慮せずに、なるべく抽象的にシステムの本質を記述する。VDM-SL によるアプリケーションの抽象仕様記述を図 3 に示す。ここでは、型定義を抽象化の基準の一つとして捉えて、抽象仕様記述で登場する概念を全て型定義で表現することにより、抽象度のレベルを示す。

図 3 では、型定義で名前、住所、および結果を定義している。名前と住所は、token 型で定義している。token 型は、VDM-SL における基本型の一つで、トークンと呼ばれる値の無限集合として定義されている。トークンとは、何がしかの値であるが、それがどのような値であるかは一切考慮していない値である。つまり、名前と住所という概念は存在するが、それらの値が具体的にどのようなものであるかは、ここでは規定しないことを意味する。コンピュータで実現する場合に、名前と住所がどのような型になるかは後で決定される。

住所録システムのシステム状態としては住所録があり、その型は名前から住所への写像型と定義している。また、operations において、「新規登録」と「検索」の機能が定義されている。次節で、ここでの抽象度と同じレベルで、ソフトウェアコンポーネントの仕様記述を示す。

```
module App

definitions

types
  名前 = token;
  住所 = token;
  結果 = <ok> | <error>

state 住所録システム of
  住所録: map 名前 to 住所
end

operations
  新規登録 (名前:名前, 住所:住所) 結果:結果
  ext wr 住所録
  post if 名前 not in set dom(住所録)
    then 住所録 = 住所録~ munion {名前 |-> 住所} and
         結果 = <ok>
   else 結果 = <error>;

  検索 (名前:名前) r:[住所] * 結果
   ext rd 住所録
   post if 名前 in set dom(住所録)
          then r = mk_(住所録 (名前), <ok>)
          else r = mk_(nil, <error>)
end App
```

図 3: アプリケーションの抽象仕様記述

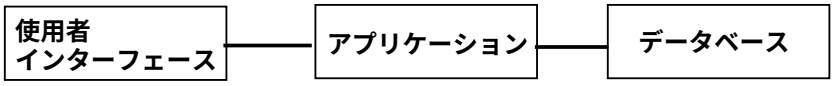

図 4: コンポーネントの構成

4.2 要求レベルのコンポーネントの仕様記述

本節では、要求レベルのコンポーネントの仕様記述例を示す。住所録システムのコンポーネントは、図 4 に示すように、(1) 使用者インターフェース、(2) アプリケーション、および (3) データベースで構成される。ここでは、VDM-SL による各コンポーネントの仕様記述例を示し、コンポーネント間の関係と、4.1 節のアプリケーションの抽象仕様記述との関連を考察する。

4.2.1 コンポーネントの記述

要求レベルのコンポーネントの仕様記述では、抽象度は抽象仕様記述と同じであるが、ソフトウェアの構造を考慮して、抽象仕様記述におけるシステム状態を、各コンポーネントにどのような責任で管理するかを記述する。まず、図 5 にアプリケーションの仕様を示す。

```
module App

imports from DB all
exports all

definitions

types
  名前 = token;
  住所 = token;
  結果 = <ok> | <error>;

operations
  新規登録: 名前 * 住所 ==> 結果
  新規登録 (名前, 住所) ==
    DB`新規登録 (名前, 住所);

  検索: 名前 ==> [住所] * 結果
  検索 (名前) ==
    DB`検索 (名前);

end App
```

図 5: 要求レベルのアプリケーションコンポーネントの仕様記述

型定義と操作定義のシグネチャは、抽象仕様記述と同じある。抽象仕様記述との違いを以下に示す。
・システム状態（住所録）はデータベースで管理するので、システム状態はない。

・機能は、それぞれデータベース（DB‘で表現）の機能を利用する。
次に、使用者インターフェースの仕様を図 6 に示す。

```
module UserIF
imports from App all
definitions
types
  名前 = App‘名前;
  住所 = App‘住所;
  結果 = App‘結果;
operations
  新規登録: 名前 * 住所 ==> 結果
  新規登録(名前, 住所) ==
    App‘新規登録(名前, 住所);
  検索: 名前 ==> [住所] * 結果
  検索(名前) ==
   App‘検索(名前);
end UserIF
```

図 6: 要求レベルの使用者インターフェースコンポーネントの仕様記述

型定義と操作定義のシグネチャは、上記のアプリケーション（App‘で表現）と同じであるが、機能は、それぞれアプリケーション（App‘で表現）の機能を利用する。また、システム状態を持たない。
次に、使用者インターフェースの仕様を図 7 に示す。データベースの記述は、システム状態（住所録）の管理をデータベースで行うので、4.1 節のアプリケーションの抽象仕様記述と同じになる。

```
module DB
imports from App all
exports all
definitions
types
  名前 = App‘名前;
  住所 = App‘住所;
  結果 = App‘結果;
state 住所録システム of
  住所録: map 名前 to 住所
end
operations
  新規登録(名前:名前, 住所:住所) 結果:結果
  ext wr 住所録
  post if 名前 not in set dom(住所録)
    then 住所録 = 住所録~ munion {名前 |-> 住所} and
         結果 = <ok>
    else 結果 = <error>;
 検索(名前:名前) r:[住所] * 結果
 ext rd 住所録
 post if 名前 in set dom(住所録)
    then r = mk_(住所録(名前), <ok>)
  else r = mk_(nil, <error>)
end DB
```

図 7: 要求レベルのデータベースコンポーネントの仕様記述

4.2.2 アプリケーションの抽象仕様記述との関係

要求レベルのコンポーネントの仕様記述は、同じ抽象レベルで、アプリケーションの抽象仕様記述を分割したものである。ここでの分割は、次節の設計レベルのコンポーネントの仕様記述を踏まえての構造を示している。抽象仕様記述は、コンポーネントの仕様とみなすこともできる。また、同じ抽象度におけるコンポーネント間の関係は、コンポーネントの「システム状態」で特徴付けられ、コンポーネント間の関係は、操作の呼び出し元も「モジュール名」と、操作の呼び出し先の「モジュール名」で特定できる。

4.3 設計レベルのコンポーネントの仕様記述

本節では、設計レベルのコンポーネントの仕様記述例を示す。データベースコンポーネントにおけるデータ型を具体化して、アプリケーションコンポーネントと関連づける。

4.3.1 コンポーネントの記述

データベースコンポーネントにおける名前と住所を文字列型と定義して、システム状態である住所録のデータ型を具体化する。VDM-SL によるデータベースコンポーネントのシステム状態の記述を図 8 に示す。

```
types                                    state 住所録システム of
  名前 = 文字列                          住所録: map 名前 to 住所
  inv s == len(s) <= 名前長さ制限;       end
  住所 = 文字列
  inv s == len(s) <= 住所長さ制限;       values
  結果 = <ok> | <error>;                    名前長さ制限: nat = undefined;
  文字列 = seq of char;                     住所長さ制限: nat = undefined;
```

図 8: 設計レベルのデータベースコンポーネントの状態記述

文字列は有限に制限される場合がある。このような制限は、名前型の不変条件 (inv) として表現できる。また、設計段階では、「制限がある」という事実が重要で、具体的にどのような値になるかは実現段階で決定されるとすれば、値として「名前の制限」を定義して、その値は未定義 (undefined) とすれば良い。

```
operations
  新規登録: App‘名前 * App‘住所 ==> 結果
  新規登録 (名前, 住所) ==
    if len 名前文字列 (名前) <= 名前長さ制限 and
       len 住所文字列 (名前) <= 住所長さ制限
    then 新規登録文字列 (名前, 住所)
    else exit <error>;

  新規登録文字列 (名前:App‘名前, 住所:App‘住所) 結果:結果
  ext wr 住所録
  post if 名前文字列 (名前) not in set dom(住所録)
       then 住所録 = 住所録~ munion {名前文字列 (名前) |-> 住所文字列 (住所)} and
            結果 = <ok>
       else 結果 = <error>;

functions
  名前文字列: App‘名前 -> 名前              住所文字列: App‘名前 -> 名前
  名前文字列 (名前) ==                         住所文字列 (名前) ==
    undefined;                                   undefined;
```

図 9: 設計レベルのデータベースコンポーネントの操作記述

名前と住所のデータ型が、長さに制限のある文字列であれば、文字列の妥当性に対して、データベースコンポーネントで責任を持たなければならない。形式的な記述では、このような責任範囲の記述が明確になる。

4.3.2 コンポーネント間の関係

設計レベルでデータベースコンポーネントを具体化したことによる、他のコンポーネントとの関係を考察する。データベースコンポーネントと直接に関連があるコンポーネントは、アプリケーションコンポーネントである。アプリケーションにおけるデータ型は、使用者インターフェースとデータベースが変更になる可能性を考慮すれば、使用者インターフェースやデータベースのデータ型に依存してはいけない。ここの設計段階では、どのような構造にするかは決定せずに、名前と住所は token 型のままにしておく。したがって、データベースコンポーネントでは、token 型から文字列型に変換する関数が必要となる。token 型ということは、型についてはまだ決定していないので、変換の関数は未定義 (undefined) である。この

後の開発工程で、アプリケーションの型が決まれば、変換の関数も定義することができる。本節の仕様記述では、関数は未定義でも、関数の構造は変わらないことが重要である。

使用者インターフェースコンポーネントは、アプリケーションコンポーネントを通して、データベースコンポーネントと間接的に関連がある。要求レベルの仕様記述では、データベースコンポーネントの結果が、そのまま使用者インターフェースコンポーネントに返しているので、データベースコンポーネントの結果が直接影響している。データベースの変更が、使用者インタフェースに影響を与えないように、アプリケーションコンポーネントの責任を明確にする必要がある。

これまで議論してきたように、コンポーネントの責任は、「システムの状態」と「データ変換関数」で特徴付けることができる。このように、形式的な仕様記述は、コンポーネント間の関係とコンポーネントの責任を明確にして、仕様を記述する人に、オブジェクトの候補の発想を支援する効果がある。

5 既存のコンポーネントの利用したアーキテクチャの記述

本章では、既存のコンポーネントを利用することを前提とした、アーキテクチャの仕様記述について議論する。例えば、使用者インターフェースとして Web の利用を前提とした、アーキテクチャの仕様記述について考える。アーキテクチャの記述は、本論文で示した (1) アプリケーションの抽象仕様記述、(2) 要求レベルのコンポーネントの仕様記述 (3) 設計レベルのコンポーネントの仕様記述について考える。

アプリケーションの抽象仕様記述

アプリケーションの抽象仕様記述は、図 3 に示した仕様記述と同じである。Web を利用したことを前提とするときに、アプリケーションの抽象仕様記述が影響を受ければ、その抽象仕様記述の抽象度は適当ではない。

要求レベルのコンポーネントの仕様記述

要求レベルのコンポーネントでは、使用者インターフェースについて考慮する必要がある。アプリケーションコンポーネントは、本稿で定義したコンポーネントを利用すべきである。このアーキテクチャでは、アプリケーションコンポーネントは、使用者インターフェースに依存しないコンポーネントとする。

使用者インターフェースは、図 5 で定義しているアプリケーションの操作を利用する。使用者インターフェースは、既存の Web クライアントと Web サーバを前提とすると、Web サーバからのインターフェースがどのようなものであるかを考えればよい。このインターフェースは、既存の Web サーバからのインターフェースで決まる。データ型は token 型のままで記述する。

設計レベルのコンポーネントの仕様記述

設計レベルのコンポーネントも、要求レベルのコンポーネントと同様に、使用者インターフェースのコンポーネントについて考慮すれば良い。ここでは、使用者インターフェースのデータ型を決定する。ここのデータ型は、既存の Web サーバからのインターフェースで決まる。

コンポーネントの関係

形式仕様記述を基に、既存のコンポーネントを利用したアーキテクチャの記述を示した。既存のコンポーネントを組み合わせてソフトウェアを構築するときに、抽象的なコンポーネントの仕様記述は、コンポーネントの再利用を検討するときに有用な情報である。形式仕様を利用して抽象度の基準を明示することが、コンポーネントの仕様を記述する上で重要である。

6　おわりに

本論文では、形式仕様記述言語 VDM-SL を用いて、コンポーネントの仕様記述例を提示して、コンポーネント間の関係について議論し、アーキテクチャにおけるコンポーネントの形式的な仕様記述の有用性を確認した。今後の課題として、
・アーキテクチャの記述の事例の蓄積と・コンポーネント間の関係の体系化があげられる。

謝辞

本研究は、「2002 年度南山大学パッヘ研究奨励金 (Pache Research Subsidy)I-A-2」よる研究助成を受けています。

参考文献

[1] D.Garlan and D.E.Perry Eds.: Special Issues on Software Architecture, IEEE Transaction on Software Engineering, Vol. 21, No. 4, 1995.

[2] J.Fitzgerald and P.G.Larsen: Modelling Systems, Cambridge University Press, 1998.

[3] 熊崎敦司, 野呂昌満, 張漢明, 蜂巣吉成: Web 情報システムのソフトウェアアーキテクチャ, オブジェクト指向 2002 シンポジウム論文誌, 2002.

[4] A.Hall: Seven Myths of Formal Method, IEEE Software, Vol.7,No.5,pp.11-19,1990.