共通問題のオブジェクト指向形式仕様記述言語Object-Zによるアプローチ

1T−1

片井　正行[1)]　田中　智[2)]　松下　芳典[2)]
1）日本アイ・ビー・エム株式会社　2）株式会社　野村総合研究所

1．はじめに

Object-Zは、形式仕様記述言語Zのオブジェクト指向への拡張である。Zは、集合論と第一階述語論理に基づく言語である。しかし、最近の問題領域の複雑化、大規模化に伴い多くの状態やオペレーションのスキーマをより容易に扱うことができる構造が必要とされるようになってきた。これがZからObject-Zへの拡張となった。[1,2]

Object-Zは、Zに対して次のような拡張がなされている。

カプセル化(Encapsulation)
・各システムコンポーネントにおける状態変数と関連オペレーションをクラスにカプセル化する。

継承(Inheritance)
・より単純なコンポーネントから仕様を追加記述する事で状態、オペレーションを含むさらに複雑な仕様の構築を容易にする。

多義性(Polymorphism)
・スーパークラスのオブジェクトが要求された時に、適したサブクラスのオブジェクトを代入させる。

また、このようなオブジェクト指向スタイルの仕様が、再利用が容易な明確な仕様を生じるだけでなく、最終的にC++などのオブジェクト指向言語へのインプリメンテーションやプロトタイプを助ける事にもなる。[3]

当論文では、このObject-Zを使って標準問題である酒屋の在庫問題[4]を解く事でObject-Zとはどのようなものなのか、さらには本当に有用なものなのかについて評価を行った。

2．評価項目

以下の４つの点について今回は評価を行った。

- ビジネスアプリケーション（伝票処理中心のアプリケーション）に対してObject-Zによる仕様記述の可能性
- オブジェクト指向の他の手法との合併の可能性
- 仕様の正確性
- オブジェクト指向への拡張による利点

3．在庫問題へのアプローチ

（１）設計例１

要件定義から直接Object-Zを適用する。要件定義からオブジェクトを切りだした後は、仕様を試行錯誤で徐々に拡張していく。

倉庫

```
containers  :コンテナ番号 ↔ コンテナ
stock       :品名 → 数量
```

Init

```
containers  = ∅
stock       = { ∀i:品名 • i ↦ 0}
```

内蔵品検索

```
req?        :出庫依頼
```

```
stock(req?.item) ≧ req?.no
```

コンテナ搬入

```
Δ(containers,stock)
tsumini?  :積荷票
```

```
∃c:コンテナ | c.init •
        containers' = containers ∪
                { tsumini?.containerNo ↦ c }
∀i:dom tsumini?.naizo •
        stock' = stock ⊕
          { i ↦ stock(i) + tsumini?.naizo(i) }
```

空コンテナ搬出

```
Δ(containers)
order?    :P出庫指示書
```

```
∀o:order? | o.mark = 空 •
  containers' = { o.containersNo } ⩤ containers
```

図−１　クラス倉庫の仕様記述例の一部

（２）設計例２

要件定義の分析においてCoad-YourdonのOOA[5]を用いてモデル化を行う。モデル化を十分行った後、Object-Zで仕様記述する。

4．考察

- ビジネスアプリケーションに対してObject-Zによる仕様記述の可能性。

・ビジネスアプリケーションを、Object-Zでも十分記述可能である事がわかった。特に、ビジネスアプリケーションでは各処理に対する仕様記述が中心であったが、Object-Zでは、事前事後状態を把握する事により、仕様がかなり正確になった。

A Case Study of Formal Specification with Object-Z
Masayuki Katai[1)]　Satoshi Tanaka[2)]　Yoshinori Matsushita[2)]
1)IBM Japan,Ltd.　2)Nomura Research Institute,Ltd.

しかし、従来のやり方に慣れ親しんだ我々にとっては、事前事後状態で物事をとらえていくやり方に慣れるにはかなり苦労をしたところであった。

● オブジェクト指向の他の手法との合併の可能性。

・今回はOOA法との併用を試みたが、OOAとObject-Zがあまり相性が良くない事が判明した。それは、OOAでのクラス構造の関係ではWhole-Part(has-a)関係として表現できるクラスが、Object-ZではPart部分を関係あるいは関数としてとらえ、状態スキーマとしてクラスの属性として取り込めてしまう場合があるからである。そのため、OOAによるモデルがそのままObject-Zに反映できないのである。

● 仕様の正確性

・酒屋の在庫問題の「コンテナ数を最小にするように搬出する」という要件についてObject-Zで表現することにより、かなり正確になる。例えば、Object-Zでは、この要件に対して、一回の搬出により空になったコンテナが最大になるようなコンテナ出庫を選択するというような仕様記述を行った。（図－２）

内蔵品出庫選択

```
Δ(containers,stock)
req?                :出庫依頼
candidateContainers :ℙℙコンテナ×出庫情報
orderContainers!    :ℙコンテナ×出庫情報
```

```
candidateContainers = { s: ℙℙコンテナ×出庫情報  |
                         s = { s1:ℙコンテナ×出庫情報  |
                            s1 = { c:コンテナ; info:出庫情報 |
                                 c ∈ ran containers ∧
                                 info.item = req?.item ∧
                                 info.no ≤ c.naizo(item) ∧
                                 info.item ∈ dom c.naizo ∧
                                 Σn:dom c.naizo •
                                   c.naizo(n) = info.no ∧
                                   info.mark = 空
                                 • (c,info)
                               } ∧
                              Σi:ran s1 • i.no = req?.no
                              • s1
                            }
                       }
∀s :candidateContainer • ∃s1:cadidateContainers •
              #{ i :ran s1 | i.mark = 空 }
                  ≧ #{ i1 :ran s | i.mark = 空} ∧
              orderContainers! = s1
∀c :dom orderedContainers! •
               containers' = containers ⊕
                                  { c.containerNo ↦ c}
               ∀i :dom c.naizo •
                       stock' = stock ⊕
                            { i ↦ stock(i) - c.naizo(i)}
```

図－２　コンテナ数最小搬出の仕様記述例

● オブジェクト指向への拡張による利点

・Classにより状態スキーマとそのオペレーションスキーマを結合させることで、より構造的になった。例えば、倉庫の場合は、倉庫の状態スキーマに対してそれを変更するオペレーションスキーマがカプセル化され、コンテナの場合はコンテナの状態スキーマを変更するオペレーションスキーマがカプセル化されるので、仕様自体は見やすくなっている（図－１）。しかし、Object-Zは、Zの拡張であるため、他のクラスの状態を直接操作できてしまい、カプセル化については不十分な点もある。（これは、Visual list[2]などで解決可能である。）

・継承、多義性については今回の仕様記述では評価しなかった。

５．課題

今回は、上流工程の評価が中心であったが、今後はさらに内部設計、実装、保守、仕様変更についての評価を行う。そのためには、さらに以下のような課題がある。

● 外部設計での結果の検証（形式推論[6]）

● インプリメンテーションするためにさらなるリファインの必要性（継承、多義性）

● 仕様記述を元にオブジェクト指向言語への実装

６．謝辞

当評価を行うにあたり、討論の場を提供していただいたYARN研究会に深く感謝する。

参考文献

[1]ISO, Z and Object-Z for use in ODP, May 1991. ISO/IEC/JTC1/SC21/WG7 N372

[2]D.Carrington, D.Duke, P.King, G.Rose, and G.Smith, Object-Z: An Object-Oriented Extension to Z, In S. T. Vuong, editor, Formal Description Techniques, pages 281-296, North-Holland, 1990

[3]R. Duke, Formal Specification of Object-Oriented systems, Tools PACIFIC'90 Sydney, 1990

[4]山崎利治、共通問題によるプログラム設計技法解説（その２）、情報処理学会, vol.25 No.11, Nov. 1984

[5]P .Coad, E. Yourdon, Object-Oriented Analysis 2nd Edition, Prentice Hall, 1990

[6]B .Potter, J. Sinclair and D. Till, An Introduction to Formal Specification and Z. International Series in Computer Science. Prentice Hall, 1991