6A-7

Object Model Transmigration 論の提案

前田　稔† 松家　英雄‡
東京学芸大学† 産能大学‡

1　従来型オブジェクトモデルの問題点

従来、オブジェクトは、コンピュータのメモリ上からの消滅でライフサイクルを終わるとされ、他のオブジェクトに生まれ変わることは想定されていなかった。このため、たとえば仕掛品の数を属性値として有する製品オブジェクトが存在するならば、「100 の状態から 150 の状態に遷移した」と表現されるように、モデル化対象の変化を状態遷移として表現してきた。その要因としては次が考えられる。

1)対象の変遷を単一オブジェクトの状態遷移としてとらえる傾向にある。
2)オブジェクトライフサイクル論は単一オブジェクトの発生から消滅までしか扱わない。
3)オブジェクトの発生に言及するデザインパターン論が少ない。
4)結合・分解関係が静的関係として扱われる。
5)複数のオブジェクトの動的連鎖関係についての記法が存在しない。

もっとも個々の仕掛品に視点を移動させ、仕掛品ごとの特性をオブジェクトで表現しようとすると問題が生ずる。たとえば「砂糖」オブジェクトと「小麦粉」オブジェクトの両者が混合加工されるとしよう。従来は、オブジェクトを消滅させる訳にはいかないので、「砂糖」と「小麦粉」の両オブジェクトを加工後も並存させた。つまり「砂糖入り小麦粉」であり、「ケーキ」の誕生ではない。しかも売れ行きに合わせて「菓子パン」の加工に変更しても、両者は同じ「砂糖入り小麦粉」である。これでは結合・分解といった製品加工がライン変更を伴いながら繰り返される様子を表せず、製品実現の全過程における個別の製品の識別と管理をオブジェクトモデル化することができない。

2　Object Model Transmigration 論の意義

Transmigration とは、死後、生と死とを繰り返すことであり、死をライフサイクルの終着点とすることのアンチテーゼとして存在する。同様の発想により、製品という枠組みの範囲内で、オブジェクトが消滅・生成を繰り返す一連のサイクルを、連続したオブジェクトモデルととらえ、これを Object Model Transmigration 論と名づけた。

Transmigration 概念の導入で、オブジェクトモデリングに関して次の利点が予想できる。

1)対象の変遷を必ずしも単一オブジェクトの状態遷移と構成しなくてもよくなる。
2)オブジェクトの消滅から発生までの間の概念上の空白を埋められる。
3)結合・分解関係について、時間軸を加味した動的関係として扱える。
4)自律的オブジェクトとして独立させることが可能になり、状態遷移ととらえることと比べ、オブジェクト部品としての交換および機能追加・削除が容易になる。
5)製品実現の全過程における製品の流れをオブジェクトモデル化することができる。
6)個々の商品オブジェクトに統一的なメソッドを用意することで、外部システムを含めいかなるシステムからも統一的なアクセスが可能となる。
7)たとえ部門ごとに異なる商品の取り扱いをしていても、オペレータにより差異を吸収することができる。
8)商品オブジェクトとオペレータとが相互にリンク情報を保持するしくみにより、オブジェクト交換や経路変更時の全体統括情報のメンテナンスが著しく軽減される。
9)システム構築前に厳密にオブジェクト粒度を定義づけなくても、オペレータによる粒度変換により、事後の粒度変換が容易になる。

3　Transmigration の要素

Transmigration 概念では、時間軸に沿って、オブジェクトが消滅、生まれ変わりを繰り返していくことを想定している。このような多数のオブジェクトからなる連鎖のなかから、構成要

Proposal of Object Model Transmigration Theory
†「前田稔・東京学芸大学」Minoru MAEDA, Tokyo Gakugei University
‡「松家英雄・産業能率大学」Hideo MATSUKA, The Sanno Institute of Management

素を抽出するならば、生まれ変わり前のオブジェクトと生まれ変わり後のオブジェクトの２つが最低限必要であることがわかる。

以上を考慮して、入力と出力を設定し、入力と出力を構成するクラスとインスタンスを組み合わせると次の表になる。

		Input1		Input2		Output1		Output2	
	type	Class	Instance	Class	Instance	Class	Instance	Class	Instance
Through	X	A	a1			C	c1		
	Y	A	a1			A	a2		
	Z	A	a1			A	a1		
Union	xX	A	a1	B	b1	C	c1		
	yX	A	a1	A	a2	C	c1		
	zX	A	a1	A	a1	C	c1		
	xY	A	a1	B	b1	A	a2		
	yY	A	a1	A	a2	A	a2		
	zY	A	a1	A	a1	A	a2		
	xZ	A	a1	B	b1	A	a1		
	yZ	A	a1	A	a2	A	a1		
	zZ	A	a1	A	a1	A	a1		
Separate	xX	A	a1			C	c1	D	d1
	yX	A	a1			C	c1	C	c2
	zX	A	a1			C	c1	C	c1
	xY	A	a1			A	a2	D	d1
	yY	A	a1			A	a2	A	a3
	zY	A	a1			A	a2	A	a2
	xZ	A	a1			A	a1	D	d1
	yZ	A	a1			A	a1	A	a2
	zZ	A	a1			A	a1	A	a1

同じクラスの場合は、同じインスタンスの場合（type-z,Z）と異なるインスタンスの場合（type-y,Y）がある。また、入力内あるいは出力内での差異に着目すること（type-x,y,z）と入力と出力での際に着目すること（type-X,Y,Z）の両者がありうる。

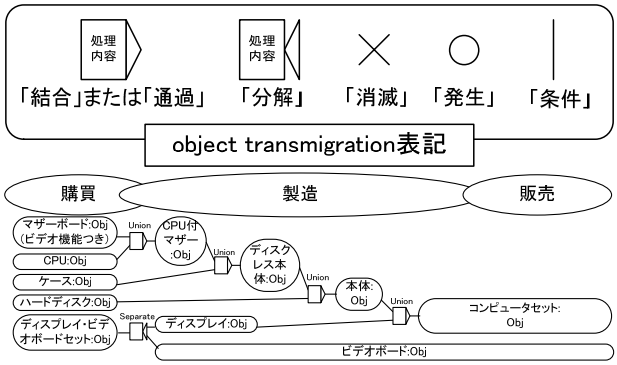
したがって、Transmigration の対象としてクラスとインスタンスを区別することが重要であることがわかる。クラスが消滅して生まれ変わることを意味するのはあくまで入力と出力のクラスが異なる場合（type-X）であり、それ以外の場合は、インスタンスも考慮して慎重に考える必要がある。

４　モデル記述方法の定義

記述に必要な要素は次の要素である。

1)オブジェクトの流れを、動的に方向性を有しつつ表現できること。
2)オブジェクトの結合と分離を表現できること。
3)オブジェクトの消滅と発生を表現できること。
4)クラス・インスタンス・メソッドの区別ができること。
5)入力と出力の数に限定がないこと。
6)生まれ変わる処理内容を記述できること。
7)生まれ変わりの条件を記述できること。
8)既存の代表的なオブジェクトモデリング表記であるＵＭＬと親和性を有するとともに、ＵＭＬ記載との間で混同・妨害が起こらないものであること。

これらを考慮して次の記述方法を定義した。



５　結合の類型

結合類型としては次が考えられる。

1)可逆型・不可逆型
2)オブジェクト保持型とプロパティ保持型
3)拡張型・縮小型
4)汎化型
5)トリガー型
6)クラス to クラス型
7)クラス to プロパティ型（個別保持型,配列型,可変クラス利用型,クラスタイプ保持型）
8)プロパティ to プロパティ型（値保持型,数値演算型）
9)メソッド型（同一移転型,縮小移転型,拡張移転型,新設型）

６　有効性の検証

生産システムの実時間監視とシミュレーションの統合を例に有効性を調査した。

手法	モデル表記	表記の直感性	データとふるまいの結合	構造変更が他に与える影響の明確性	製品種類ごとの独立性	個々の製品ごとの独立性
1)データベース＋構造化	エンティティ関係図	△	×	×	△	△
2)オブジェクト従来型	標準UML(ステートチャート)	△	△	△	△	×
3)Object Model Transmigration	Object Transmigration表記	○	○	○	○	○

１）の場合は、固定的な構造で生産システムの実時間監視を行うことは容易であったとしても、複数の構造を仮想的にシミュレートしたり、シミュレーション結果にあわせて構造を柔軟に変更したりすることが難しい。２）の場合は、状態の変化は属性の変化となり、ふるまいの変化までは表現できない。また、多数の状態を含めようとすると、広い範囲でオブジェクトを定義せねばならず、柔軟性を有しない。

３）の場合は製品の組み換えが容易である。個々の製品に必要なだけの情報とふるまいを有しており、製品の場面ごとに必要な情報を格納でき、オブジェクトが巨大化しない。そして、隠蔽により外部との結合情報が限定されており、結合方法の標準化により組み換えが容易になる。このような柔軟性は生産システムの実時間監視とシミュレーションの統合に適すると思われる。