『UML モデリング教科書 UML モデリング L1 第 2 版』補足

株式会社テクノロジックアート
テクニカルデプト　照井　康真

書籍『UML モデリング教科書 UML モデリング L1 第 2 版』で説明しきれなかった内容や、実際のソフトウェア開発で活用するためのポイントなどの補足情報です。なお、事前に書籍をお読みになっていることを前提としておりますので、まだお読みでない方は、是非、ご参照ください。

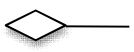
1. 集約関係とコンポジション

「クラス間の関係を定義する上で、集約関係とコンポジションの使い分けがわからない」といった声をよく耳にします。これは、以前の版である UML1.x では、集約関係とコンポジションの違いが、今現在の版よりも曖昧であったことが、原因のひとつとして考えられます。

1-1. 集約関係とコンポジションの違い

集約関係とコンポジションの違いに重点を置いた説明は、次のとおりです。

表 1 集約関係とコンポジションの説明

No.	項　目	表　記	英　名	日本語直訳	説　明
1	**集約関係とコンポジション**に共通した内容	集約関係かコンポジションのいずれか	aggregation	集約	オブジェクト間の関係が**「全体」と「部分」の関係**であることを表す。「has-a」関係や、「is-part-of」関係とも呼ばれ、全体側のオブジェクトは、部分側のオブジェクトを**所有している**と言われたり、部分側のオブジェクトは、全体側のオブジェクトの**一部である**と言われたりする。
2	**集約関係**固有の内容		shared aggregation	共有集約	下の「項目No.3 コンポジション（composite aggregation）」に比べ、**より弱い**集約の関係。部分側のオブジェクトが、同時に複数の全体側のオブジェクトから**集約される**ことがあり得る。全体側のオブジェクトが破棄されても、部分側のオブジェクトは、破棄された全体側のオブジェクトとは別のオブジェクトの一部であるため（または、将来的に一部になることがあり得るため）、**通常、同時に破棄されない**。 例えば、「得意先」と「仕入先」オブジェクトは、ひとつの「取引先」オブジェクトを共有していると捉えることができる。これは、ひとつの取引先が、得意先にも仕入先にも、同時になり得るからである。仮に、その取引先が仕入先ではなくなり、仕入先オブジェクトが破棄されたとしても、得意先としては依然として存在するため、取引先オブジェクトは破棄されない。なお、これらの、得意先、仕入先、取引先間の関係は、集約関係だけが唯一の表現方法というわけではなく、多重継承や動的分類などによって表現する場合もある。

3	**コンポジション**固有の内容		composite aggregation	合成集約	上の「項目№2 集約関係（shared aggregation）」に比べ、**より強い**集約の関係。部分側のオブジェクトが、同時に複数の全体側のオブジェクトから**集約されない**。つまり、部分側のオブジェクトから見て、全体側のオブジェクトの数は、0（存在していない）か、1（ひとつだけ存在する）のどちらかであり、2 以上にはならない。部分側のオブジェクトから見て、唯一の全体側のオブジェクトが破棄されると、その時点で一部となっている部分側のオブジェクトも、**通常、同時に破棄される**。これは、「ライフサイクルが同じ」と言い表されることがある。 例えば、「本棚」に収納されている「本」は、同時に複数の本棚に収納されることはない。本棚から出されているか、どれかひとつの本棚に収納されているかの、いずれかである。仮に、本棚に本が収納されている状態のまま本棚を処分すると、本棚に収納されている本も同時に処分される（本棚を処分する際には、あらかじめ本棚から本を取り出しておくことをお勧めする）。なお、これらは、あくまでも物理的な観点のみから見た場合の関係であり、論理的な別の観点を優先して考慮する必要がある場合などは、必ずしも本棚と本がコンポジションの関係になるとは限らない。図書館の蔵書管理システムを例に取ると、本棚が処分される時に、その本棚に収納されているはずの本が、本棚と同時にどうなってしまうかは、通常、考慮しないことが多い。たとえ、本が収納されている本棚が新しいものに買い替えられようとも、本が図書館の蔵書であることに影響を与えないからである。この場合は、本が、図書館や目録といった、本棚とは別のオブジェクトとコンポジションの関係にあると捉えることもできる。

1-2.　集約関係とコンポジションの使い方

UML の仕様で定められているのは、あくまでも表記法だけであり、モデリング方法は、プロジェクトにとって扱いやすいものを選択することができます。今回の題目では、クラス図を書くときに、必ずしも集約関係とコンポジションを区別して書かないといけない、というわけではないということです。ある集約関係が、厳密にはコンポジションなのかどうか、といったことを考慮しない方が、都合が良い場合もあります。

次の表は、集約関係とコンポジションの使い分けという観点で、代表的な使い方を、大きく 6 つに分類したものです。

表 2 集約関係とコンポジションの使い方

	(a) 集約関係とコンポジションを区別しない場合	集約関係とコンポジションを区別する場合	
		(b) 集約関係と通常の関連を区別する場合	(c) 集約関係と通常の関連を区別しない場合
(1) 分析モデリング	**(1a)** 「全体」と「部分」の関係を、**すべて集約関係**として記述する。つまり、クラス間の関係として、通常の関連と集約関係は登場するが、コンポジションは登場しない。部分側のオブジェクトを表す概念が、複数の全体側のオブジェクトを表す概念から共有されるかどうかや、ライフサイクルが同じかどうかまでは、**設計的にだけではなく、概念的にも考慮しない**。システムに関係がある概念構造のみを明確にするという、**一番の目的に集中**することで、モデリング作業の効率化が期待できる。	**(1b)** ひとつのクラス図の中に、**集約関係とコンポジションの両方**を混在させる。つまり、クラス間の関係として、通常の関連と集約関係、そしてコンポジションが登場する。概念的に、部分側のオブジェクトが、複数の全体側のオブジェクトから共有されないといった**ビジネスルールを、クラスの構造で明示したい**場合などの使い方。明示されたビジネスルールは、システムの仕様として、入力値の検査や、操作時の動作などに反映される。	**(1c)** 左の 1b の**集約関係を、通常の関連**に置き換える。つまり、クラス間の関係として、通常の関連とコンポジションは登場するが、集約関係は登場しない。これは、集約関係とコンポジションの違いほど、集約関係と通常の関連に違いがないことによる。集約関係と通常の関連を区別しなくとも、システムの仕様には、ほとんど影響を与えない。むしろ、**あいまいな情報を排除**することによって、関係者間のコミュニケーションが円滑化されることを期待できる。
(2) 設計モデリング	**(2a)** 上の 1a の場合と同様に、「全体」と「部分」の関係を、**すべて集約関係**として記述する。多くのプログラミング言語において、コンポジションは、集約関係や通常の関連と同様に、構造的には、オブジェクトの属性として扱われる。このため、実装上のオブジェクトが、共有されるものなのかどうかや、オブジェクトが破棄されるタイミングといった**仕様は、必要であれば、シーケンス図などの別の図や、文章、制約などで記述**する。	**(2b)** 上の 1b の場合と同様に、ひとつのクラス図の中に、**集約関係とコンポジションの両方**を混在させる。実装上のオブジェクトが、共有されるものなのかどうかや、オブジェクトが破棄されるタイミングといった**仕様を、クラスの構造で明示したい**場合の使い方。例えば、XML の中で、あるオブジェクトの情報が、別のオブジェクトのタグの中に含まれるのか、それともただ単に参照されるだけなのかなどの違いを表現できる。	**(2c)** 上の 1c の場合と同様に、左の 2b の**集約関係を、通常の関連**に置き換える。多くのプログラミング言語において、集約関係と通常の関連には、実装上の違いはない。**効果が少ない作業を排除**することによって、設計作業の時間短縮が期待できる。

このように、ひとことで集約関係とコンポジションと言っても、使い方はひとつだけではありません。また、今回紹介した使い方以外の使い方もあるかもしれません。集約関係とコンポジションに限らず、プロジェクトで最適な使い方を選択することが、UML の長所を引き出し、ソフトウェア開発を円滑に進めるポイントです。