6A-4

クラス図に埋没する構造の管理とその応用の提案

石井俊直* 山岡孝之† 堀池聡‡
三菱電機 (株) 先端技術総合研究所

1 はじめに

ソフトウェア開発においてパターンは重要な解決策を提供する。UML エディタでは、GoF のデザインパターンなどがあらかじめ用意されていて、それを編集中のクラス図に取り込んで使うようになっているものが多い。パターンの適用は品質の高いソフトウェアの設計に結びつくばかりではなく、ソフトウェアの再利用の意味のおいても重要な役割を持つ。MDA に代表されるモデル駆動型ソフトウェア開発では特に、単に設計者がパターンを用いるというだけではなく、開発ツールが成果物におけるパターンの存在を認識していることが、ソフトウェア再利用にとって重要になる。この観点からは、UML エディタにおいてデザインパターンを取り込んでモデリングを行うことができるというだけではパターンは設計過程における初期値として使われているというだけであり、充分とはいえない。

本報告では、一般的な UML エディタを構成するインターフェイス要素に、構造化ビューを付け加えることで、クラス図の編集においてパターンと成果物を、設計者にとっても設計ツールにとっても、より緊密に結びつけるための方法を提案する。

2 ソフトウェア再利用とパターン

OMG が標準化を進める Model Driven Architecture (MDA)[1] では、実現しようとするものをプラットフォームに依存しない形でモデル (PIM) に表し、これを実現のためのプラットフォームに依存するモデル (PSM) に変換することで開発を進めるとされているが、このプラットフォームが何であるかについては、必ずしも明確な定義がなかった。これに対して、Atkinson ら [2] は、プラットフォームとは、言語と、あらかじめ用意されたタイプとインスタンス、そしてこれらを利用するルール、すなわちパターン、の四つの側面から定義できると提案している。このモデルの特徴は、プラットフォームの指定に実行基盤といった概念が必要ではないことにある。プラットフォームを定めるためにパターンが本質的な役割を持つことから、モデルをプラットフォーム間で変換することでソフトウェア開発を駆動する MDA のスタイルでは、パターンは、単に設計者が質の高い成果物を作るための解決策に収まらない。パターンはモデル変換における本質的情報で、その認識なしに変換は困難となることが示唆される。また、必ずしもモデル変換に関わらなくても、成果物にパターンあるいはモデルに特徴的な構造をツールが認識することで実現される有用な機能は他にも考えられるだろう。

パターンを UML で表す場合、図1 のように、パターンをコラボレーションとして表すことが多い。この記法は説明用に使われ、実際の設計でこうした書き方をすると設計した部分と説明用の部分の区別がつかないため、実際の設計で用いることは難しい。設計者も設計ツールもパターンの存在を明確に認識することが重要であることを考えると、この状況は望ましいとは言えない。

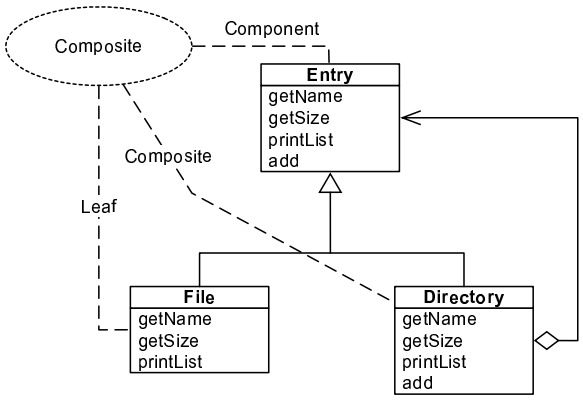

図 1: クラス図にパターンを明示する方法

この問題を解決するため、次節で、まず、一般的な UML エディタのインターフェイスの構造をレビューする。

A Method of Managing Semantic Structure Embedded in Class Diagrams

*Toshinao Ishii・Advanced Technology R&D Center, Mitsubishi Electric Corp.
†Takayuki Yamaoka・Advanced Technology R&D Center, Mitsubishi Electric Corp.
‡Satoshi Horiike・Advanced Technology R&D Center, Mitsubishi Electric Corp.

3 UMLエディタのインターフェイス

UMLクラス図を編集するために様々なツールがあるが、そのインターフェイスには共通した構造が見られる。図2にこれを示した。その主要部品は、Resource View、Diagram Edit、Meta-Model Paletteと示した三種類のインターフェイスである。

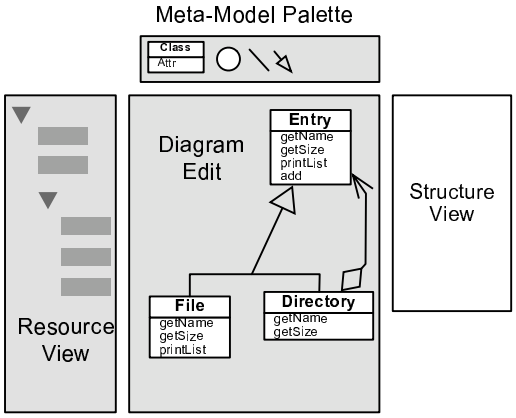

図 2: 灰色で示したResource View、Meta-Model Palette、Diagram Edit編集の三つが一般的UML編集ツールによく見られるインターフェイスパーツである。Structure Viewが本論で提案する四番目のインターフェイスパーツ。

Resource Viewは、おもに成果物にアクセスするためのインターフェイスである。ツリー配置にすることで多くの成果物要素へのアクセスを容易にしている。Meta-Model PaletteはUMLメタモデルに定義されたものをダイヤグラム部品としてインスタンス化することを容易にしている。Diagram Editインターフェイスは、作業対象のダイヤグラムを表示/編集するためのエリアである。ダイヤグラム編集作業はこの他に、選択されたインスタンスの詳細情報を表示するウィンドウや、コンテキストに応じてポップアップしてくる種々のインターフェイスを使いながら行われるが、その中にはAtkinsonら[2]が指摘した、用意されたタイプやインスタンスを取り込むためのものも多く見られる。

インターフェイスの構造を見ると、ダイヤグラムとパターンを区別する方法が提供されていないことがことがわかる。

4 構造化インターフェイス

先に述べたように、パターンやイディオムなどの形で現れる解決策が開発成果物に埋め込まれている様子は、重要な関心事である。そこでこの情報を扱うための第四のインターフェイスを提案する。これが図2のStructure Viewである。

その実装は様々考えられるが、直感的には図1のパターンに相当する楕円がStructure Viewに表示され、さらに何らかの方法で構造要素とダイヤグラム要素との関係を示す機能があれば良い。無論、この関係を示すことができるのだからツールがその関係を認識している。

構造要素とダイヤグラム要素の関係を表すには、一般的なUMLエディタの動作パターンを応用することができる。例えば、ダイヤグラム要素を選択すると、プロパティ表示ウィンドウに選択要素の情報が自動的に表示される動作パターンを応用して、選択されたダイヤグラム要素が含まれる構造要素がハイライト表示されるといったものが考えられる。逆に構造要素を選択すると対応するダイヤグラム要素がハイライトされるというものも考えられる。

関係の識別ばかりではなく、モデル要素の構造化作業を行う場合にも、インターフェイスが独立していることを利用できる。図2では、構造と要素が明確に区別されるので、その対応を指定する操作が容易になる。例えば、ある構造要素が選択された状態でモデル要素をインスタンス化するとう操作で、テキスト的な設定を行うのに比べより直感的にモデル要素を特定の構造の部分として生成できる。

5 まとめ

モデル駆動型開発では成果物に埋め込まれるパターンの存在を開発ツールも認識する必要が生じる。提案したUMLエディタの第四のインターフェイスは、設計者のパターンを通した設計と、ツールよるパターンの認識に有効と考えられる。

参考文献

[1] David S. Frankel, *Model Driven Architecture*, Wiley, 2003

[2] Colin Atkinson and Thomas Kühune, *A Generalized Notion of Platforms for Model-Driven Development*, in Model-driven Software Development, ed. S. Beydeda, M. Book and V. Gruhn, pp. 119-136, Springer Verlag, 2005