2C−6　構造化オブジェクトモデリング環境 SOME の拡張 -UML 表記の編集機能と SOMM 表記との相互変換機能の追加-

田渕 和幸　原田 実

青山学院大学理工学部経営工学科

1. はじめに

我々は、これまでオブジェクト図を階層的に構造化した表記法を用いる構造化オブジェクトモデリング環境 SOME の開発を行ってきた。今回は SOME に UML 表記のオブジェクト図の編集機能を追加し、SOMM 表記と UML 表記の相互変換機能をもたせた。

2. SOME／UML

2.1. SOME

SOME は図 1 のようにオブジェクト図を階層的に構造化した表記法を用いている。このことは設計図の巨大化／複雑化を避け、クラス間の集約関係を階層関係にすることで直感的な理解を促進している。またクラスがそのまま設計単位となっていて自然なモデリングを可能にすると共に、実装工程へのスムーズな移項を可能にしている。図 1 は「表計算問題」のモデルの一例で、クラス Hyoukeisan に Uriagefile、Hyougoukei、Goukeifile という 3 つのクラスが集約されている。

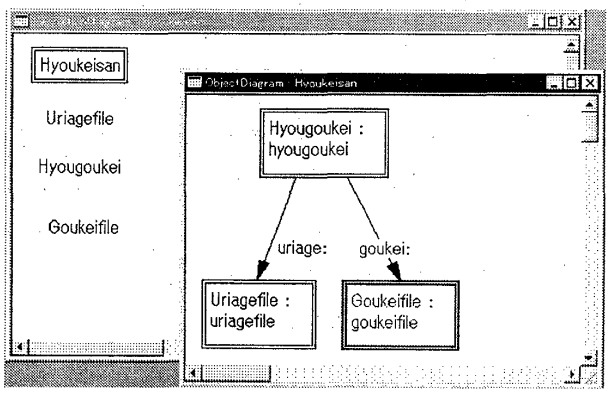

図1　SOMM表記の派生図(左)とオブジェクト図(右)

Enhancement of Structured Object Modeling Environment SOME- augmentation of UML edit function and the conversion function to SOMM-
Kazuyuki Tabuchi, Minoru Harada
Dept. of Industrial and Systems Engineering, College of Science and Engineering, Aoyama Gakuin University

Hyougoukei : hyougoukei は前者がクラス名で、後者がオブジェクト名となっている。

2.2. UML

図 2 は実際に SOME に追加した UML 表記のオブジェクト図のエディタである。またこの図は図 1 の SOMM 表記を変換したものである。一般的に知られているこの UML 表記は、オブジェクト内の 3 つの関係(関連関係・派生関係・集約関係)を 1 枚の図にの中に表現することにより「設計情報の一覧的理解」というメリットを持っている。

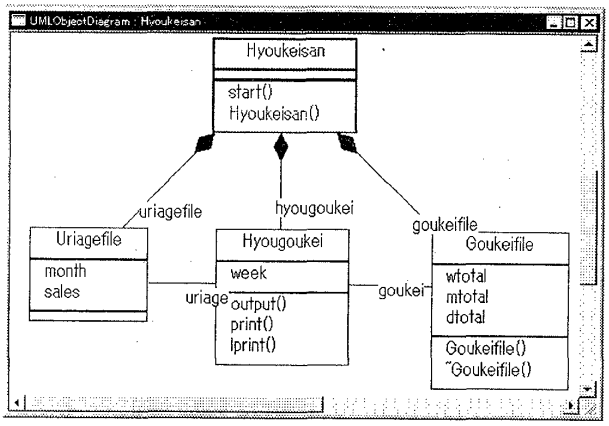

図2　UML表記のオブジェクト図

2.3. 相互変換機能

同じ設計図であっても SOMM 表記と UML 表記では視覚的に全く違うものになってしまう。本研究ではこの表記法の違いを誰もが簡単に且つ速やかに変換できることを目指した。しかしおもに 3 つの大きな問題がでてきた。

まず第 1 に、片方の表記法が必要とするデータを他方の表記法が持っていなかった場合の変換である。とくに SOME では自動的なコード生成を行えるよう詳細な情報を持たせるようにしていたが、UML 表記では意識的にその情報は持たせていない場合が多い。具体的には図 3 のようなコードを設計したい場合それぞれの表記は図 4 のようになる。C++プロ

グラムを自動生成する為にはc1、c2といった変数名（オブジェクト名）は必ず必要となる。SOMEではオブジェクト単位でグラフノードを作成している為にオブジェクト名はC:c1のように必然的に表記されている。UMLではクラス単位でグラフノードを作成する為にオブジェクト名を意識的に表記されることは少ない。本研究のUMLではオブジェクト図のC:c1と記述されている後者のc1をロール名として記述することにした。つまり図3のようにクラスCに集約関係のアークを複数ひきそれぞれにc1、c2というロール名を記述させるようにした。

```
Class C  {     .....
         }
Class A  {
           C c1;
           C c2;
         }
```

図3 コード例

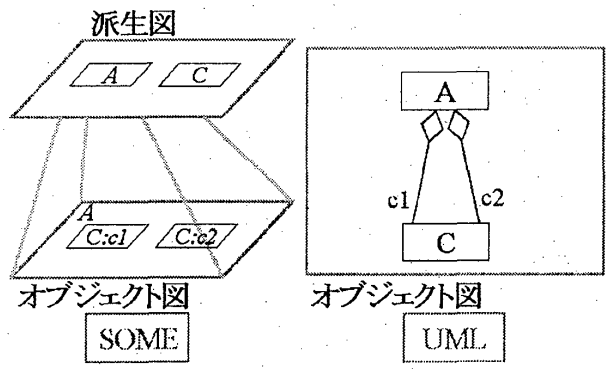

図4 図3のコードに対するSOMM表記とUML表記

第2に、変換のタイミングである。初めは両方のエディタで編集を行っていた場合、片方を編集するたびに他方も随時編集されていくよう考えていた。しかし「随時変換」を行っていると、間違った編集を行っていても変換してしまうというデメリットや、両方がメモリを随時使用する為無駄なメモリの使用や編集操作の遅れに繋がるというデメリットが発生してしまう。そこで変換機能は「一括変換」という考え方で進め、ユーザーが変換ボタンを押すと同時に全てを変換するようにした。これによりメモリ不足の心配がなくなりユーザーが望むときに適切な変換を行えるようになった。

第3に、変換後のノードやアークの位置である。ノードにしてもクラス関係のアークにしても、異なる表記のエディタに移ると、変換前のエディタの位置のままではクラス同士、アーク同士が重なりあってしまう。つまり、変換すると整っていた設計図も入り組んだ設計図になってしまう。それを改善する為に考え出したのが「自動再配置」という機能である。この機能はノードの重なりやアークの必要以上の重なりを自動的に整理し、文字通り再配置する機能である。またユーザーが動かしたくないノードをピンで留め、固定して「再配置」を行うことも可能にした。

以上のように処問題を解決し、本研究の相互変換機能を実現した。

2.4. 設計図記述言語DSL

SOMEは、様々なオブジェクト指向CASEツール間においてその設計図やデータの相互利用を図るために、設計図記述言語DSL形式のテキストファイルで設計情報を外部保存する。本研究でUML表記の設計図の格納にも設計記述言語DSLを用いるようにした。更にSOME、UMLともに入出力時刻が記憶され、DSLにおけるSOMEとUMLの混合を防ぎ、一目で入出力時刻が分かるダイアログも開発した。

3. おわりに

一般的に知られているUML表記を使用できるようになったことでより多くのユーザーがSOMEによるシステム設計を行えるようになった。様々な設計に対してSOMM表記とUML表記を使い分け、作業中に相互変換することが可能になり、より効率的で正確な作業を行えるようになった。そしてその両方の表記法から自動的にC++プログラムのコード生成を行うことのできるSOME／UMLが実現された。

4. 参考文献

[1] 岩田 隆史他：Windows環境で動作する新・構造化オブジェクトモデリング環境SOMEの開発研究、青山学院大学 卒業論文(1996)