ソフトウェア工学 102-15
（1995．1．26）

ソフトウェア情報モデルとパターン

友枝 敦*
tomoeda@itc.co.jp

（株）情報技術コンソーシアム
〒135 東京都江東区木場5-11-13 木場公園ビル

ソフトウェア再利用の分野では、近年パターンに注目が集まっている。筆者は、ドメインの分析が問題解決への近道との立場から、ソフトウェアに関する情報の定義，モデル化を行った。その結果、ドメインあるいはタスク固有の抽象的アーキテクチャ（とその活用法のセット）として、自然にパターンのモデル化も行われたので報告する。また、パターンに関しては未だ研究者間での共通認識は得られてはいないが、その中でも比較的著名なP.Coad,E.GammaそしてC.Alexanderのパターンと筆者のソフトウェア情報モデルとの関連に関しても言及する。

キーワード：オブジェクト指向，再利用，パターン，抽象アーキテクチャ，ドメインモデル

Object-Oriented Information Model on Software and Patterns

Atsushi Tomoeda
tomoeda@itc.co.jp

Information Technology Consortium
Kiba-Kouen Bldg. 5-11-13, Kiba, Koutou-ku, Tokyo 135, Japan

Recently "pattern" has been regarded as a helpful concept to realize software reuse. This paper describes patterns found in an abstract architecture specific to certain domain and tasks. These patterns are by-product of our current research on domain analysis which, we believe, is a key to successful software development. The paper also compares our software information model with patterns suggested by P.Coad, E. Gamma, and C. Alexander. Their works are well known in the field of software pattern researches where new theories have still been vigorously introduced.

key words : Object-Oriented, reuse, pattern, abstruct architecture, domain model

* （株）エヌ・ケー・エクサより出向中

1　はじめに

ソフトウェア再利用の意義は多くの人が認めるところであり[1]、オブジェクト指向がこの問題に対する有力な解決策となるのではないかと期待されている[2]。だが現在提案されている主要なオブジェクト指向開発方法論は、オブジェクト指向技術の持つモデル記述力に焦点を当てた新規開発のための方法論である[3][4]。過去のソフトウェア開発により得られた知識／情報を再利用することに軽く触れているものはあるものの[5]、既存の知識／情報の再利用を前提としているものは見受けられない。

一方、OOP(Object-Oriented Programming)の分野では方法論とは状況が異なり、Smalltalkに代表されるクラスライブラリがそれなりに成功を納めている[6]。確かにこれらを利用すれば多大な恩恵を得ることができるが、その享受のために支払う投資は決して少なくはない。クラスライブラリは、その大半が基本部品など汎用的なボトムアップ部品からなっており、次の要求を利用者に突きつける。これらをクリアしても、実際にクラスライブラリの威力が発揮されるのは専ら設計フェーズ以降となる。

- 部品の識別／利用法
- フィロソフィの理解
- 分析モデルからフィロソフィへの適合変換
- クラスライブラリ間のコンフリクト対応

特に上位２つが、半日で修得できる簡単な文法を持つSmalltalkが難しいとの評価を受ける一因である。C++も今後、オブジェクト指向のメリットを求めれば求めるほど、Smalltalk以上の投資が必要となる可能性が高い。

上記、クラスライブラリに於ける現状の問題を打開するために、GUIなどの領域での成功に見られるようなトップダウンアプローチ[7]，いわゆるより上流からのソフトウェア情報の再利用が試みられている。トップダウン的な部品は、特定の領域の問題解決の解法を、クラスライブラリのフィロソフィーに適合する形で部品化したものである。すなわちトップダウン部品は、状況に応じたボトムアップ部品の組合わせルールを包含していると見ることができる。従ってトップダウン部品を利用することで、前述の基本クラスライブラリ利用にまつわる投資，問題点の削減が期待できる。

この様な観点から、近年ソフトウェア再利用の枠組みとしてトップダウン的にして、より上流のソフトウェア部品であるパターンに期待がよせられている[8]。だが、パターン研究はその途についたばかりであり、未だパターンについての研究者の間でも共通の定義，認識も得られていないのが実状である。

本研究はソフトウェアに関する各種情報の再利用を目指したものである。まず筆者は、ドメインの分析が問題解決への近道との立場から、ソフトウェアに関する情報の定義，モデル化を行った[9]（以降このモデルを“ソフト情報モデル”と略称する）。その結果、抽象的なアーキテクチャ（以降、“抽象アーキテクチャ”。アーキテクチャは各種モデルやコードなど実現，実装の総称）として、パターンも自然にモデル化がなされた。本稿では、ソフト情報モデルならびにその中でのパターンの位置付けに関し議論する。まず２章で再利用の観点からオブジェクト指向の進化を考察する。次に３章では本研究の基盤となるソフト情報モデル及びその中でのパターンの位置付けを解説する。そして４章でソフト情報モデルと現在著名な幾つかのパターンならびに関連技術との比較，関連について述べる。最後に５章でまとめを行う。

2．オブジェクト指向の進化

再利用に関して、オブジェクト指向は、現在最も有望な解であると判断するものの、上述の如く未だ満足のいくものとは言いがたい。筆者はかねてより、上述のモデル記述，クラスライブラリ，パターンへと続く動きは、図１に示すようなオブジェクト指向の進化発展の流れに基づくものと考えている[9]。

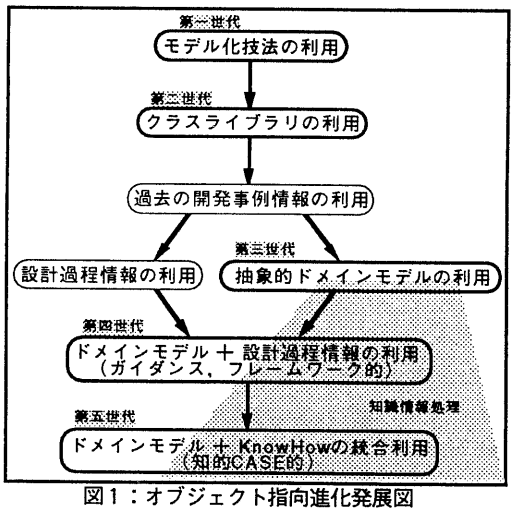

図１：オブジェクト指向進化発展図

今後ソフトウェア再利用にさらなるブレークスルーを起こすためには、SmalltalkのMVCモデルに代表されるような、ドメインあるいはタスク固有の解法モデル（ドメインモデル）ならびにその活用に関する知識の取扱が大きな鍵となる[9][10]。ドメインモデルはそのドメインに共通する抽象アーキテクチャとしてあらわされ、柔軟で広範な適用を可能とする。優秀なソフトウェア技術者は、この抽象アーキテクチャを多く保持しており、この知識／情報の存在が、彼らの直感力に優れた問題解決能力の源と考えられる[11]。

これら抽象アーキテクチャとその活用法が、データベースとして、あるいは活用／開発支援環境，フレームワークとして実現されるのが第四世代である。抽象アーキテクチャとその活用法が形式的／知識ベースとして取り扱われ、知的CASEとして実現されたのが第五世代である。ここでは人間の持つ知的な知識利用の様に、類似のものを認識／流用した問題解決も実現されることが望ましい。この抽象アーキテクチャ（とその活用法）こそが、筆者の考えるパターンであり、第三～五世代を視野に入れたものである。

3．ソフト情報モデル

筆者が提案するソフト情報モデルを図２に示す。ソフトウェア開発は、様々なレベルの「要求や仕様（以降、“仕様”）」を元にそれを具現化するためのアーキテクチャすなわち「実現構造（以降、“構造”）」を導き出す作業、を再帰的に繰り返すものと捉えられる。そして各々の「構造」毎に、その「仕様」からの開発の歴史、すなわち「プロセス」に関する情報が存在する。この「仕様」を基点とした、「構造」に「プロセス」が絡んだ三組モデルが、ソフト情報モデルの基本形である[9]。

「仕様」や各種「構造」は「プロダクト」に関するものである。この２つは「プロダクト」の状態，表現／Viewを表している。「Stack」の「仕様」、そして「リストによる実現，実装」，「アレイによる実現，実装」などのように。この例から分かるように、通常一つの「仕様」には、複数の解法，すなわち「構造」が対応する。そして各々の「構造」は物理的な特徴を有し、これが同じ「仕様」に対応する他の「構造」との間に性能差となって現れる。ソフトウェア開発では、置かれた状況に適した「構造」を設計／選定する事が肝要となる。

オブジェクト指向の場合、「ソフト情報」はオブジェクト及びサービスに関するものとに大別される。一般論的に、メソッドの「構造」はオブジェクトの「構造」に依存する。「オブジェクト」はさらに「ドメイン」，「タスク」，「ベース」に、同様に「サービス」は「プリミティブ」とその拡張サービス「エンハンス」に分類される。

一方の「プロセス」情報は、「設計／開発ログ」から、「仕様」と「構造」との対応を記述した「マッピング」情報、そしてその文脈中で得られた知見まで、様々な「仕様」－「構造」間の情報が記述される。

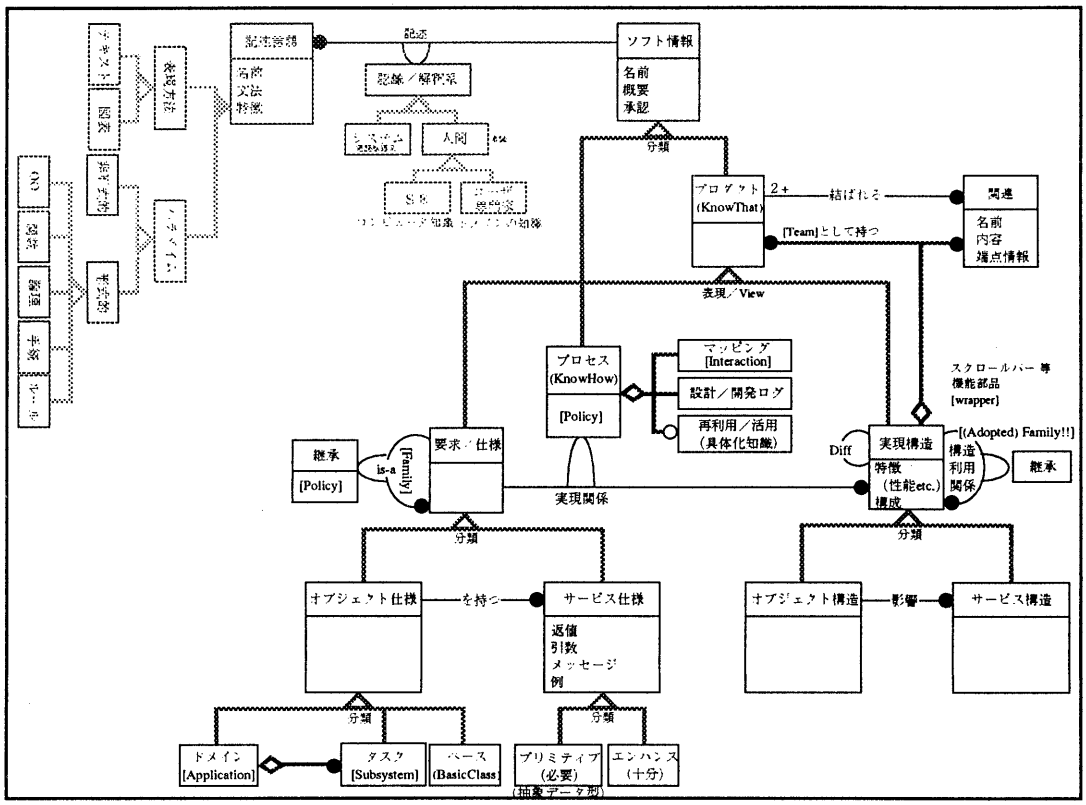

図２：ソフトウェア情報モデル図　-OMT オブジェクトモデル

「仕様」の記述例を図3に示す。「仕様」は「ソフト情報」の存在を示すノードと捉えられる。図書館などのシステムレベルからIntegerや加算などのプリミティブレベルまで様々な粒度が存在する。内容記述は一般に非形式的，人間向けに書かれる。

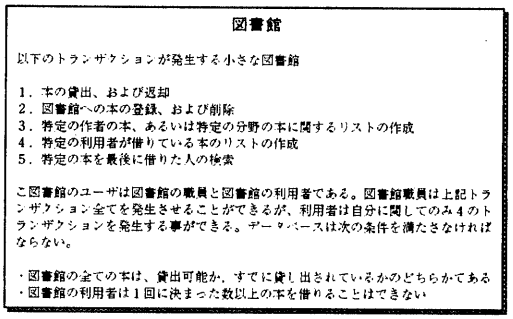

図3：『仕様』例　―図書館―

同様に単純化した「構造」の例を図4に示す。実際にはこれに性能等に関する記述が付加される。「構造」の構成要素であるオブジェクトやサービスもまた「ソフト知識」である。また「構造」は、分析モデルから設計モデルやコードまで様々な記述，詳細レベルが存在する。これらのレベルを決定付けるのは、一般に記述言語とその解釈，承認系がどの程度の記述量／詳細度を求めるかによる。詳細な考察はなされていないが、先述の「記述言語」の言語要素は「オブジェクト」の「ベース」サブクラスとなるであろう。

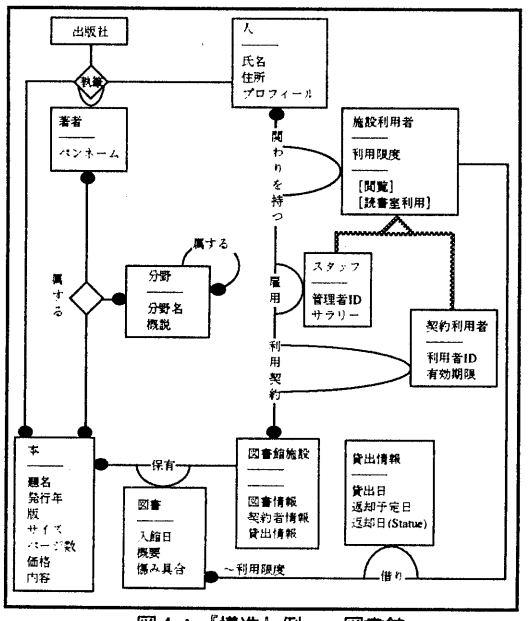

図4：『構造』例　―図書館―

さて、再利用に於いて、継承は非常に重要な概念，技術である。だが近視眼的な流用志向の継承がまかり通るのは、再利用にとっては悪しき風潮である[9]（表1参照）。流用志向によって形成されたクラス階層は、偶発的なプロダクトの開発，保守の歴史を表しているに過ぎない。この様な論理性一貫性を持たないクラス階層は、言わば、うかつに足を踏み込むことの出来ない密林の様なものである。さらには知識情報処理への進化をも阻害するものである。

表1：継承分類表

関係		概要	上位との差異	
			仕様	構造
概念的上下関係		通常の is-a 関係。抽象具体，汎化特殊化など概念的な上下関係が存在。	有り	具体化有り
拡張／修正進化／退歴関係 (VersionUpに関連)	機能／仕様向上	機能追加，拡張等に関係。	小	小
	性能向上	性能改善を目指したもの。最適化指向。	無し	大
	バグフィックス	バグの修正を施したもの。	無し	小
	OS等の違いに対応	OSなどの変更／バージョンアップに伴う変更。	概ね無し	概ね有り
類似／流用関係		概念的上下関係の成り立たない単なる部分的流用を目指したもの。	小中	小中

ソフト情報モデルではこれらを次の様に取り扱う。

- 仕様に差異　→　「仕様」の階層
- 仕様は同じ構造に大差　→　実現関係
- 仕様は同じ構造に小差　→　「構造」の階層

これにより「仕様」の階層は概念的なものを、「構造」の階層は解法などの流用を、それぞれ分離して取り扱うことができる[9][10]。尤も「仕様」の階層から「構造」の階層へ強い関連があることは想像に難くないが。

継承を取り扱う際には、継承の単位，何を継承させるのかも重要な問題である[10]。図5に図書館の上位概念である貸著作物屋の「仕様」例を示す。図書館との差異は斜め太字で表されている。両者の違いに関係した名詞（オブジェクト），動詞（サービス）である“図書館”，“図書”，“読書室利用”が、各々その上位概念である“貸著作物屋”，“著作物”，“サービス室利用”に置き換えられていることが分かる。同様に「構造」例を図6に示す。「構造」では「仕様」に比べさらにブレイクダウ

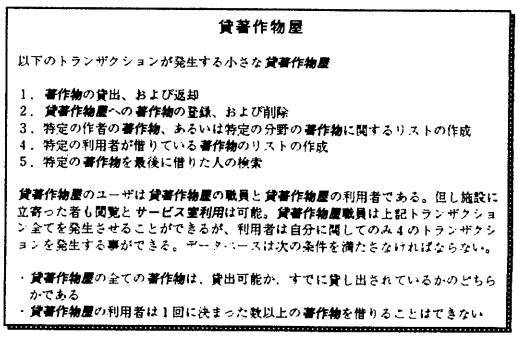

図5：『仕様』例　―貸著作物屋―

ンされた感がある。様々なオブジェクトや関係そして属性やサービスまで様々なものが置き換えられている。

これらの例により、『仕様』，『構造』によって継承させるべき単位，要素が異なることが分かる。ここから継承単位は記述言語依存であることが予想される。さらにこれらは、一次元的な属性の継承から二次元的なアーキテクチャの継承への進化の重要性を示している。

ところで、貸著作物屋の例は、抽象アーキテクチャ，すなわちパターンの簡単な例ともなっている。『仕様』の抽象化に伴い『構造』も抽象化され、そのアーキテクチャを構成するオブジェクトが抽象化される。この様に抽象的なオブジェクトを含むモデルが、“抽象アーキテクチャ”である。図7に模式的にパターン例を示す。薄墨は上位からの継承を、白抜は抽象的であることを表している。太線で示した貸レコード屋の開発を例に考えてみよう。貸著作物屋は2つの異なる「構造」を持っている。まずは性能面で現在の要求に近い貸著作物屋「構造」を選ぶ。次いでその構成要素の内、抽象的な部分を各々具体化すれば良いのである。具体化に関しては「プロセス」情報がガイド，ヒントを提供してくれる。

パターンに於いては、前述の三組みモデルも抽象度が上がり、ドメイン，ドメインモデルとその活用情報となる。最も抽象度の高いトップレベルパターンはソフトウェア全般に関するものであり、そのプロセス知識はソフトウェア工学的手法やOMTなどの方法論となる[9]。

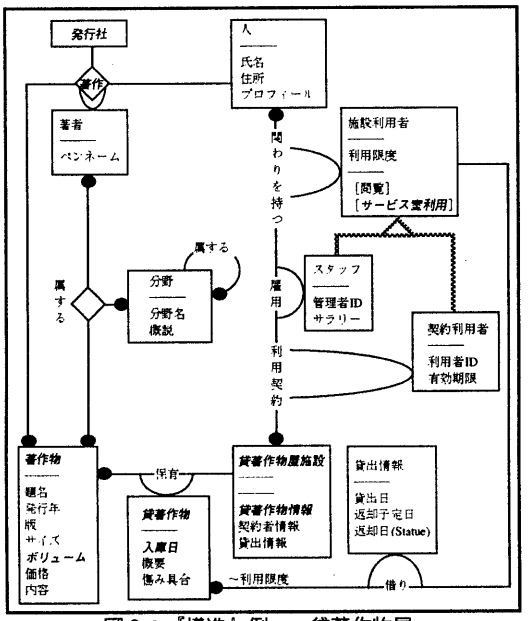

図6：『構造』例　－貸著作物屋－

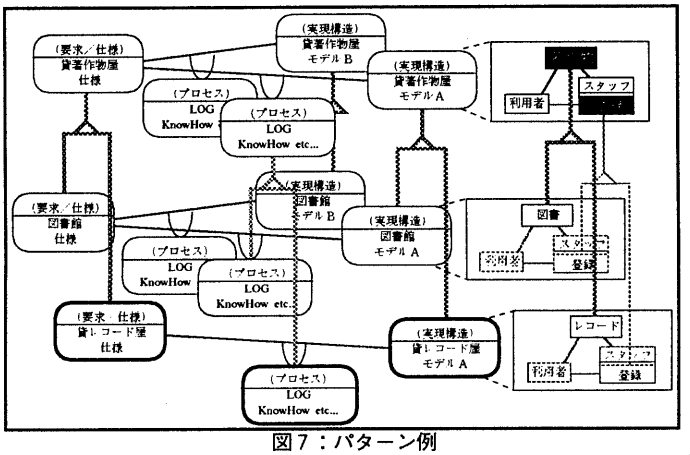

図7：パターン例

4．他のパターン、関連技術との比較

現在著名なパターンそして知識情報処理などの関連分野とソフト情報モデルとの比較，関連を表2に示す。

現在著名なパターンであるP.Coad, E.Gammaのパターン[8][12]は、形（構造）からのアプローチであり、結果的に汎用のボトムアップ的部品の範疇を抜け出してはいない。いわば第二世代の洗練されたものと位置付けされる。その影響もあり、パターンの具体的用法が貧弱である感は否めない。P.Coadは論文[4]で、パターンとは分子のようなものだと結んでいる。筆者の考えるパターンは様々なレベルの柔軟な問題解決子であり、材料と作り方が記述してあるレシピの様なものだ。従って分子というよりは、むしろ遺伝子に近い。

現在のパターン研究の火付役となったC.Alexanderのパターン[13]は、ドメインを意識したトップダウン的部品であり、第三，四世代に属する。ただ残念なことにパターンの構造が明示的に示されていない。ソフトウェア分野での応用のためには更なる分析設計が必要である。

一方、知識情報処理などの分野との比較では、『仕様』をLHS(Left Hand Side)，『構造』をRHS(Right Hand Side)と見立てた、TRS(Term Rewriting System)との対比もおもしろそうである[14]。また、『仕様』を結果，『構造』を原因と対応させ、類推機能を持たせるのも期待の持てるアイデアである[10]。知識表現上では、『仕様』はWhatを表し，『構造』はどの様に実現したかというHowを表す。とは言え『仕様』はソフト知識の存在を示し、その内容がHowで代弁されていても構わない。また、複数「構造」の中から一つを選ぶ選定理由として、各「構造」の持つ特徴，性能要件がWhy，根拠として示される。『プロセス』は、どの様にして「仕様」から「構造」が作られたかと言うHow、そしてなぜその様な

表2：ソフトウェア情報モデルと他パターン，関連技術との比較表

他モデル / 本モデル	Pattern			Knowledge-Based					
	C.Alexander Model	E.Gamma Model	P.Coad Model	表現	知識深さ	TRS	P.H.Winston 類比	ProblemDomain	
要求／仕様	問題	[ ] に対応	タスクに相当	What	浅い	LHS	結果	設計↓ 条件，制約 説明	診断↑／解析
プロセス	解決 文脈 解決の適正な選定，具体化をもたらすための状況，制約			How／Why	説明／深い				
実現構造				How／Why	深い	RHS	原因		

Term Rewriting System → Object Rewriting System!!

「構造」になっていったかと言う判断の歴史Whyを表す。『仕様』→『構造』という方向は設計問題を表すが、その逆は診断，解析問題として、保守，デバッグそしてリバースなどへの応用が期待される。

5．おわりに

現在まで、個々のパターンのモデルは幾つか提案されている。本稿ではこれらとは一線を画し、ソフトウェア開発に関する情報のモデル化を通じ、パターンのメタなモデルを提示した。これにより本ソフト情報モデル中で、他の著名な研究者のパターンも統合的に位置付けされた。さらには、知識情報処理などとの融合による今後の応用にも期待がもたれるモデルになっている。

とはいえソフト情報モデルは曖昧，抽象的な部分を残しているのも事実である。今後はこれらの点を明確化するためにも、ソフト情報モデルを基にしたパターン活用ツールの試作へとフェーズを移して行きたいと考える。

最後に、P.Coadが論文[8]であげた疑問に答えることで今後の課題と展望に代えさせて頂くことにする。

（1） システム的なパターン発見が可能か？

パターンマッチ，抽象化や類推の取扱いなど知識情報処理の導入が必要。ソフト情報モデルを基にした基本的なアイデアはあるので、別の機会に議論する。

（2） パターンには階層があるのだろうか？

もちろん。クラスに階層がある様に。パターンは特別なものではなく、感覚的には抽象クラスを実現するアーキテクチャに過ぎない。だがこれは重要な進化である。

（3） 例を元にどうやってガイドラインを導くか？

抽象化と関連する知識常識を用いて。プロダクト情報のみではプロセス情報を導き出すのは困難と思われる。

（4） パターンとパターンを結びつける戦略は？

『構造』例で見たように、パターンはその組合せ方も含め、自身の中にソフト情報（子パターン）を含んでいる。性能要求も実現関係の上で「仕様」と「構造」（パターン）とを結びつける。これ以外の連携については、図4が一つのヒントを与える。すなわち、『人』や『本』は図書館の存在には依存しない独立なものであるが、これらが図書館と関わった時、『施設利用者』等関連オブジェクトとして表れてくる。これらは『図書館』と外界をつなぐI/Fの役割を担う。C/Sなどソリューションに関するパターンとの融合は困難が予想される。

（5） いつパターンが関係パターンと結び付くか？

4より想像される様に、子パターン，仲間パターン共に、自身の発生時に関係づけられるのが理想。相手がまだ存在していない場合は、自身と同時に生成する。実現関係の構築も、性能要求など文脈に基づいた適切なパターン（「構造」）の連結を暗示している。

謝辞

本研究は、情報処理振興事業協会(IPA)の研究の一環として行われた「オブジェクト指向によるソフトウェア生産技術開発」プロジェクトによるものである。技術アドバイザとしてご指導いただいた東京大学の玉井哲雄教授，北海道大学の田中譲教授を始めとする関係者の皆様に感謝いたします。

参考文献

[1] 千吉良英毅,小林正和:"ソフトウェア再利用技術の動向",人工知能学会誌,Vol.2 No.3,pp.316-323

[2] Cox,B.J.他著,松本正雄訳:"オブジェクト指向のプログラミング",トッパン,1992

[3] Rumbaugh,J.他著,羽生田栄一監訳:"オブジェクト指向方法論OMT",トッパン,1992

[4] Coad,P.,Yourdon,E.著,羽生田栄一監訳:"オブジェクト指向分析(OOA)[第二版]",トッパン,1993

[5] Booch,G.:"Object-Oriented Analysis and Design with Applications, Second Edition", The Benjamin/Cummings Pub.Co.,Inc.,1994

[6] Pinson,L.,Wiener,R.著,羽生田栄一監訳:"Smalltalk:オブジェクト指向プログラミング",トッパン,1990

[7] Weinand,A.,Gamma,E.,"ET++ - a Portable, Homogenous Class Library and Application Framework"

[8] Coad,P.: "Object-Oriented Patterns", Commun ACM, Vol.35 No.9 pp.152-159,1992

[9] 友枝:"オブジェクト指向を拡張したソフトウェア知識表現の試作",やわらかなソフトウェアに関する調査研究報告書",JISA/JSD, 1992

[10] 友枝:"機能と構造とを分離した知識表現形式とそのソフトウェア知識への適用事例",日本電気基幹技術研修論文集,1988

[11] 青木 淳著: "オブジェクト指向システム分析設計入門", SRC, 1993

[12] 藤野晃延,佐藤啓太:"クラスライブラリにおける設計ノウハウの共有と再利用",SEA ソフトウェアシンポジウム in 函館,1994

[13] Alexander,C著. 平田翰那訳:"時を超えた建設への道",鹿島出版会, 1993

[14] 竹内寛,佐藤明良,清沢治彦:"SOFTEX:ソフトウェア設計エキスパートシステム(3)—変換システム—",第43回情報処理学会全国大会,1991,10