ソフトウェアの抽象化指向特性と特許

東京大学
玉井哲雄

1995 年 3 月 10 日

1　はじめに

ソフトウェアは他の工業製品とは異なる特性があり，それが特許法による法的保護を馴染まないものにしていることは，既にいくつかの視点から議論されている．たとえば，新規性・進歩性の判定が困難なこと，陳腐化が早く審査に要する時間の長さと適合しないこと，特許による公開で模倣がしやすくなるが，侵害の認定はむずかしいこと，などである [5, 6].

ここではソフトウェアあるいはアルゴリズムの本質的特性と思われる，抽象化・一般化の指向という観点から，特許との関係を考えてみたい．まず，第 2 節でプログラムとアルゴリズムの抽象化とはどんな概念であり，どれほど重要な特性であるかを述べる．次に第 3 節で，1993 年のコンピュータソフトウェアの法的保護に関する国際会議で取り上げられた例題によって，ソフトウェアの特許出願に際して，なるべく登録が認められるように記述するという努力は，この抽象化・一般化とはまったく逆の方向に向かってなされるものであることを示す．以上を踏まえて，第 4 節で議論を行い意見を述べる．

2　プログラムとアルゴリズムの抽象化

ソフトウェア技術の歴史は，なるべく広い範囲で適用可能な一般性の高い技術を探求するものであったといえる．そのためのもっとも基本的な方法論・枠組みは，抽象化の機構であった．ソフトウェアを構成する構成要素，すなわち手続き，データ，オブジェクト，それらで構成されるサブシステム，さらには全体のソフトウェアシステム自身に至るまで，その抽象度が高いほど利用価値も高い．

以下で，そのような抽象化の機構の例を具体的にみてみる．

2.1　プログラムの抽象化

2.1.1　手続きの抽象化

プログラムをまとまった機能単位の手続きに分解することは，プログラミング技術の比較的初期に考え出された工夫である．その単位は，プログラミング言語によって，サブルーチン，プロシージャなどと呼ばれた．

このような手続き単位は，単にシステムを分解する際の構成要素となるだけでなく，手続きの抽象化を可能とするものでもある．手続きを外部からみた場合，必要となるのはどのような入力を受け付け，それに対してどのような出力を行うかという機能仕様であり，その機能が具体的にどう実現されているかという内部構造は無関係である．このことは手続きという仕組みが，内部構造を捨象することによって，抽象化機構として働きうることを意味する．

同じような機能を実現している手続きでも，その機能範囲がどれほど一般化されているかという基準により区別が可能である．たとえば，与えられたデータの列を一定の順序で並べ替えるソートという手続きを，考えてみよう．単純なソート手続きとしては，0から9までの1桁の数字を昇順に並べ替えるものが考えられよう．それに対し，任意の桁の整数の列を並べ替えることができる手続きの方が，より一般性が高い．実数も並べ替えられれば，より一般的である．さらに順序が定められるのは数値とは限らない．文字をアルファベット順に並べる，単語を辞書式に並べる，なども同様の並べ替え操作である．並べる順序も，昇順か降順か選べる方が一般性が高い．

このように手続きの一般化を進めてより利用範囲を広くするということは，抽象度を高めることに相当する．整数でも実数でも文字でも文字列でも対象とすることができ，また大きい順にも小さい順にも並べ替えられるというのは，数学的にいえばそれらのデータの順序性という性質を抽象化し，そのように抽象化された性質を持つデータ一般に対して手続きを適用可能にするということである．具体的に整数をデータとして与えるのか，文字列をデータとして与えるのかは，手続きを呼び出すときに指定すればよい．これは手続きに与える入力として，単なるデータだけでなくデータの種類（型）を許すという考え方である．このような指定のための入力をとくにパラメータということがある．

別の例をあげてみよう．ある関数，たとえば誤差関数を，与えられた区間で数値的に積分する手続きがあるとする．積分の対象となる関数をより一般化して，入力として指定できるようにした方が利用範囲が広いことは明かである．数学の関数もプログラムでは手続きとして実現される．したがって，積分を行う手続きに，非積分関数の手続きを渡してやる機構が存在すればよい．いくつかのプログラミング言語では，このように手続きをパラメータとして手続きに渡したり，手続きの出力として数値でなく手続きそのものを与えることができるものがある．この場合も，手続きの機能をより一般化するために，手続きへの入力や出力を抽象化するという方法をとっていると解釈できよう．

以上の議論から，手続きを一般化し抽象化することで，その利用範囲が拡大し，利用価値が高まるという命題が成立することが明らかに言えるだろう．

2.1.2 データの抽象化

手続きの抽象化にやや遅れて，データの抽象化という概念が1970年代以降発展し今日に至っている．プログラミング言語には通常，整数，実数，文字などの基本的なデータの型が定められている．抽象データ型とは，このようなデータ型の概念を拡張して，より抽象度の高い複雑な構造を持ったデータの型を，自由に定義して使えるようにする仕組みをいう．

たとえば待ち行列という抽象データ型を考えてみよう．待ち行列とは，データをしまっておく場所で，いつでもデータをつけ加える，また取り出すことができるが，取り出すときはもっとも早く入れられたデータから順番に出される，というものである．ここで行われている抽象化は，まず待ち行列というデータ構造を，データを入れるという操作と取り出すという操作の組としてとらえ，さらにその関係として先入れ先出しという性質を規定することで定義して，その内部構造には立ち入らないという方法を取ることによっている．データ構造の具体化には，より抽象度の低い，配列あるいはリストというデータ構造が使われることになるが，抽象データ型の仕様としてはそのデータ型に適用可能な操作の組とその性質を定めれば十分である．この関係は，手続きに対して外からは機能仕様がわかれば十分であるという意味での抽象化と対応している．

待ち行列にどんなデータを入れられるかで，抽象データ型としての種類が異なってくる．たとえば整数を入れる待ち行列，文字列を入れる待ち行列，さらに複雑な構造を持ったデータを入れる待ち行列，などが区分される．しかしこれも，抽象データ型の定義にパラメータを導入することで一般化できる．すなわち，データの種類である整数，文字，などの情報をパラメータ化す

ることにより，さらに抽象度の高いデータ型の定義となる．この関係も，ソート手続きを例にとって説明した，手続きにおけるデータ型のパラメータに対応するものである．

2.1.3 オブジェクトの抽象化

上に述べた手続きの抽象化とデータの抽象化は，たがいにまったく独立したものではない．手続きの機能を一般化するために，定められた性質を抽象化したデータのクラスに適用するという考え方を見たし，データの抽象化はそれに対する操作手続きの集合を定義するという枠組で実現できることを述べた．ある意味で，この手続きとデータの抽象化の概念を統合したのが，オブジェクト指向技術における抽象化であると見ることができる．

オブジェクト指向技術は 80 年代から注目を浴び始め，問題分析・設計からプログラミングにいたるまで，その適用分野を拡大しながら発展を遂げつつある．オブジェクトとは，まとまった意味をもち，あるいは一定の動作を行なう“もの”や概念を指す．オブジェクトは，操作の対象となるデータの構造を抽象化したという抽象データ型に近い静的な側面と，自らの状態をもち，外界とのやりとりによって動作する状態機械を抽象化したという動的な側面とをもつ．したがって，手続きの抽象化やデータの抽象化で述べたようなパラメータ化という仕組みは，自然に導入される．

オブジェクト指向技術では，さらに抽象化をより明示的に扱う方法をもっている．それは抽象化によるクラス階層という概念である．クラスとは同じ属性・動作をもったオブジェクトの型を指す．クラス階層の上位にあるものはより抽象的，一般的なクラスであり，下位にあるものはより特殊化されたクラスである．下位のクラスは上位のクラスの属性・動作を継承した上で，さらに独自の属性や動作を加えてもつ．この継承という機構が，単に概念の整理だけでなく，プログラミング技術上も仕様やコードの共通部分の重複を省き，再利用を促進するという効果をもたらす．また階層構造で整理されたオブジェクト・クラス群は，種々の分野で共通ライブラリとして提供され，広い範囲に流通しつつある．

多くの場合，提供される抽象度の高いオブジェクトを，利用者が自分の構築するシステムの目的にしたがって特殊化・具体化して利用するという形態がとられる．ここでもより抽象度の高いオブジェクトの方が利用範囲が広いという一般論が成り立つが，さらにその抽象度の階層構造全体が与えられることにより，さらに利用価値が高まるといえる．

2.2 アルゴリズムの抽象化

プログラムはアルゴリズムを特定の記述言語で記述したものと見ることができるから，プログラムの抽象化の議論はそのままアルゴリズムの抽象化の議論としても成り立つ．しかし，ここではアルゴリズムの対象とする問題分野を考慮した，アルゴリズムの内容に立ち入って考えてみたい．

たとえばネットワーク上の最適化問題として古くから研究され応用例も多いものに，最短経路問題がある．ある点から別の点へ，ネットワーク上の経路をたどってもっとも短い距離で行ける方法を求めるものである．まったく異なる分野であるプログラミング言語のコンパイラ技術で，データフロー解析と呼ばれる問題分野がある．最短経路問題とデータフロー解析問題は，一見まったく無関係に見えるが，適当な数学的抽象化を行なうと両者が同じ構造を持った問題として定式化できるだけでなく，その解法アルゴリズムも共通な形に一般化できることがわかる．さらにこの形式に帰着できる問題は多く，たとえば通信ネットワークの信頼性解析問題，有限オートマトンの解析，高速自動微分法などが同じ枠組で扱え，一般化したアルゴリズムが適用可能である [2].

このように異なる分野で独立に研究され開発されたアルゴリズムが，抽象化・一般化するこ

とにより本質的には同じものであることが後から発見される例は数多くある．たとえば伊理による最近の報告で，材料力学分野における Mohr の応力円という技法と，確率分布論における離散分布の平均と標準偏差が満たすべき拘束条件を求める方法とは，線形計画法の問題として定式化すると同じ構造となり，解法として共通な構造を持っているという事実は，その見事な一例である [1].

このような状況をまとめると，次のようになろう.

1. ある問題分野でアルゴリズム A が発見され，それとまったく別の分野でアルゴリズム B が独立に発見された.

2. A と B とが本質的に同じ構造をしていることは，自明ではない．しかし問題の構造に洞察を加え，その本質を抽象化することによって，2 つのアルゴリズムが根本的に同等であることが，後から示された.

この場合，次のようなことがいえよう.

- アルゴリズムの利用者としては，一般化されたアルゴリズムを知ることは有益である．問題の本質がよく理解できる上，さらに別の分野にも適用可能かも知れない.

- 一般化されたアルゴリズムがそのままプログラムとして実現され，それを具体化することにより個別の分野に適用可能となると，さらに有用である． 2.1 節で述べたようなプログラムの抽象化の枠組を用いると，一般化アルゴリズムの実現と，その具体化による適用が技術的に可能となる.

3 例題による事例研究

SOFTIC Symposium '93 (The Fourth International Symposium on Legal Protection of Computer Software) において，ソフトウェア特許の問題を考える上での例題として出された 2 つのうち，数値の並べ替え (sorting) をテーマとするものを取り上げ，クレームの記述方法と特許性の関係を見る.

3.1 ソートを使った例題

この例題には 6 つのクレームの記述例が含まれる [7].

1. クイックソートが仮に新たに発明されたものであるとして，そのアルゴリズムをメモリ内の配列に対する操作として記述したもの．その操作は「数値データ列を (ある) 境界値以下の値をもつデータ群と，境界値より大きい値をもつデータ群とに分かれるように転送する」というかなり抽象度の高い記述になっている．また，再帰的な繰り返し方法と終了条件も，同じレベルの記述である.

2. 1 と同じクイックソート処理を，プログラムレベルで記述したもの．操作の対象はメモリ内の配列で 1 と同じだが，数値群を初めの位置と終りの位置を指すポインタの対で表し，数値の入れ換え操作もポインタの移動を用いて具体的に表現している．また，再帰を具体的に実現するために，スタックというデータ構造を明示的に用いている.

3. 3 以降でもソート処理がテーマだが，クイックソートという指定もないし，ソート方法に関する技術的な記述はない． 3 では物品に記述された住所を配達の順序にしたがって並べ替えてプリントするという，ソートの応用を記述したものである．住所と配達番号の対応

表を持ち，入力された住所データ群から配達番号を引いて，その配達番号順にソートして出力する.

4. 3 と同じ処理だが，入力にイメージリーダーを用い，文字認識処理を使って文字コードに変換したものを住所データとする，というところが異なる.

5. 4 に加えて物品を 3 つのスタッカを用いて物理的に搬送する仕組みを想定し，配達番号順のソート結果にしたがって，物品自身を再配列するという方法を記述したもの.

6. 4 と同じ内容を方法ではなく装置として記述したもの.

このうち，1 から 5 までは方法の特許としての請求であり，6 のみが装置の特許請求となっている.

3.2 審査判断の比較

この例題に日米欧の特許庁担当官が判断を示している．その結果は表 1の通りである.

表 1: 日米欧の特許庁担当官による判断

	日本[1]	米国[2]	欧州[3]
1	×	×	×
2	△	×	○
3	×	×	×
4	×	×	○
5	○	○	○
6	△	○	○

○: 特許性あり，×: 特許性なし，△: 場合による[4]

この議論はパネル討論形式でなされたが，この結果について他のパネリストから，米国は普段より厳しく，欧州は普段より緩い判断になっているのではないか，という意見が出された．また，他のパネリストでこの 6 つのケースについての個々の判断を発表したものもあったが，いずれもさらに広く特許性を認めるもので，1 を除き 2 から 6 までは特許性ありというもの，さらに 1 も含めてすべて認めるというものすらあった.

3.3 特許性判断の基準

6 つのクレーム例を改めて比較してみる．1 と 2 を比べると，明らかに 1 の方が抽象度が高い．専門的な視点から見ると，1 の記述はややあいまいであるが (そのあいまいさを図解例で補っている)，同じ抽象度のレベルでより厳密な記述を行なうことは可能である．ただし，厳密さを

[1]相田義明 (日本特許庁)

[2]S. G. Kunin (The Patent and Trademark Office)

[3]A. S. Holzwarth(The European Patent Office)

[4]日本の 2 の判断は，アルゴリズムからの自然の帰結という意味では×だが，具体的な規定になっているという意味では○とも判断されるというもの．また同じく 6 は，4 と変わらないという意味では×だが，アルゴリズムの占める割合が低いという点では○とも判断されるとする.

追求してなんらかの形式的な記述言語を用いて書いたとしたら，現状ではクレーム記述として受け入れられないだろうが．2は1のインプリメンテーションの細部を記述しているが，このレベルでの実現方法には，他にもいくつかのバリエーションが考えられる．

3–6は住所を配達番号順に並べ替えるという同じ応用例を扱っている．ソート技法としてクイックソートを用いるとした方が例題としての斉合性がとれるように思うが，3–6ではソート方法は特定していない．3はコンピュータにデータを直接入力し，結果をプリンタに出力するというもっとも単純な利用法である．4はイメージリーダーを付加し，5はさらにスタッカという物理的な装置を用いたシステムに組み入れるということにより，物理的な機器との結びつきを順に高めている．6は4の記述方法を変えただけである．

ソフトウェアあるいはアルゴリズムという観点からいえば，3–6の本質的な部分は3の記述で尽きている．3以外はハードウェアの要素を付加したに過ぎない．

1–2と3–6を比べると，3–6はソート技法を特定していないので技術内容がないともいえるが，仮に3–6もクイックソートを前提としていたとすると，これは1–2の適用対象を特定したものといえる．

以上をまとめると，3種類の対比ができる．

1. 抽象化 (1) 対 具体化 (2)

2. 一般化 (1,2) 対 特殊化 (3,4,5,6)

3. 純粋なソフトウェア (3) 対 ハードウェアの付加 (4,5,6)

そして，このいずれの軸も特許性の審査判断と強く相関があり，より具体化，特殊化されたもの，さらによりハードウェアとの結びつきが強いものが，特許を認められる傾向にある．このことは，表1で示された判断結果からも明らかであるが，さらに日本のコンピュータ・ソフトウェア関連発明の審査基準や，米国における数学的アルゴリズムを記載するクレームに対する2段階テストの考え方からも，自然に導かれる．

4 議論

第2節でやや詳しく述べたように，ソフトウェアやアルゴリズムは，常により抽象的，一般的な表現を求めて進歩してきた．より抽象的，一般的に記述されたソフトウェア / アルゴリズムは，適用範囲が広くまた柔軟で，その意味で利用価値が高い．このことは，より具体的，特定的な記述が意味がないということでは決してない．しかし，具体的，特定的な記述が，より抽象的，一般的な記述と関係づけられ体系化されることで，きわめて強力な技術となるということがいえる．

一方，第3節で見たように，ソフトウェアやアルゴリズムの特許は，より具体的で対象の特定化された記述を優先し，さらにハードウェアとの結びつきが強いものを優先する．このことから，次のような問題が生じるだろう．

1. 汎用性の高い有益なアルゴリズムやソフトウェアを考案したとしても，特許出願をする際には，わざわざ具体化・特定化した表現に変え，またあえて装置と結びつけたクレームとする必要がある．広い具体化の技術や適用範囲を確保しておきたい場合は，その一々について出願しなければならない．

2. 限定された分野の限定された手法がすでに特許化されている場合，それを含む汎用的で強力なアルゴリズムやソフトウェアを発明したとしても，その特許性に問題が認められる可能性が低い．

3. ある分野に対し適用されるアルゴリズムが公知であるかすでに特許化されている場合，別の分野に類似の手法を適用したものの特許性の判断がむずかしい.

4. 一般化，抽象化されたアルゴリズムが公知の場合，それを具体化し特定の分野に適用するという方法や装置が特許出願された場合に，その特許性の判断がむずかしい.

ソフトウェアの特許に関連して議論される多くの問題が，ソフトウェアの本質としての抽象性・一般性・論理性と，特許出願に求められる具体性・特定性・物理性という矛盾から生じている．たとえば，次のようなものである.

均等論　均等論とは，文理解釈上は特許請求の範囲に含まれないが，請求範囲に記載された発明と均等な発明を技術的範囲に属するものとして，権利範囲を拡張的に解釈する手法だという[4]．ソフトウェアのインプリメンテーションで特定のデータ構造や技法を使ったものをクレームとして記述した場合，そのインプリメンテーションの仕方を変更したものに権利範囲が及ぶかという問題や，あるいは適用分野を変えるという上記3の場合の判断に関連するが，この議論の有効性についてはまだ定説がないようである.

間接侵害　装置としての特許請求の方が認められやすく，また方法の特許は権利行使の範囲が方法の使用行為に限定されることから，ソフトウェアを装置の一部として特許出願されることが多い．その場合，同等の機能を持つコンピュータプログラムは，間接侵害として解釈するという考え方があるようである．これも特許は物理性，特定性を求め，ソフトウェアは論理性，一般性を持つことからくる，解釈上の問題である.

機能クレーム　一般に機能のみを記述したクレームは，具体的な実施方法が明確でないため特許性がないと判断される．しかし，ソフトウェアの場合，機能か方法かという判定は相対的なものである．抽象度の階層から見て上のレベルの記述は，その下のレベルの操作を単に「機能」として用いる記述となろう．しかし，全体としては，その抽象度のレベルに対応した「方法」の記述になっているはずである．その境界は一概に決められない．プログラム自体が特許の対象とはならないために，特許請求の記載にプログラムを添付する代わりにフローチャートを添付することが行なわれるようである．しかし，ソフトウェア技術からするとプログラムに対応するフローチャートというのは，過去の遺物である．それが特許出願ではまだ使われるところに，機能クレームの解釈の問題が如実に出ている.

このような問題に対して，まったく逆向きの2つの考え方がありえよう.

- これまでの具体性，特定性，物理性を要求する審査基準を抜本的に改めて，抽象的，論理的なクレームでも特許性がありうるものとする.

- ソフトウェアやアルゴリズムと特許は本来的に馴染まないものとして，原則的にそのような特許を認めない.

前者の考え方は，上記の問題のうち1を解決するが，2～4については大いに問題が残る．2については，後から出願されたより一般性のある方法の特許が認められうるかも知れないが，それとより具体的な記載内容の特許との関係がどうなるのか問題である．3についてはなんら解決にならない．4については，一般化，抽象化されたアルゴリズムが公知の場合だけでなく，さらに特許が認められた場合が，新たな問題として加わる.

したがって，筆者の意見は，後者を支持せざるをえないというものである．汎用性の高いアルゴリズムやソフトウェアを発明したものは，それで利益を得る道があることが望ましいと思

う．抽象度や一般性が高いアルゴリズムは価値が高いということも，すでに指摘した通りである．しかし，そのような抽象的，論理的クレームに対して特許を認めることにより，もともと想定すらされていない範囲にまで及ぶ広い権利保護を与えてしまうことの弊害は，きわめて大きい．かといって，現在のようにあえて装置に結びつけ，具体的，特定的に記述されたもののみが特許を与えられるということも，ソフトウェアの本質にまったく反するものである．となると，ソフトウェアやアルゴリズムに対する権利保護は，やはり著作権などの方法に頼らざるをえないのではないだろうか．

参考文献

[1] 伊理正夫: Mohr の応力円，確率分布，線形計画法，応用数理， Vol. 4, No. 4 (1994), pp. 4–14.

[2] 玉井哲雄：グラフ上の一群の不動点問題とその逐次型解法，電子情報通信学会論文誌 A, Vol. J76-A, No. 1 (1993), pp. 45–53.

[3] 豊田正雄：ソフトウェアと特許権，ダイヤモンド社， 1992.

[4] 光主清範: コンピュータソフトウェアの特許侵害について — 特許法及び新審査基準に基づく権利解釈から —，特許研究， No. 17, 1994 年 3 月， pp. 39–49.

[5] League for Programming Freedom: Against Software Patents, *Communications of the ACM*, Vol. 35, No. 1 (1992), pp. 17–22, 121.

[6] Samuelson, P.: Should Algorithms be Patented?, *Communications of the ACM*, Vol. 33, No. 8 (1990), pp. 23–27.

[7] The Proceedings of SOFTIC Symposium '93 — The Fourth International Symposium on Legal Protection of Computer Software, Software Information Center, November 1993.