第12章　順序集合分割法【その1】
アクティブ・オブジェクト・モデリングの方法論

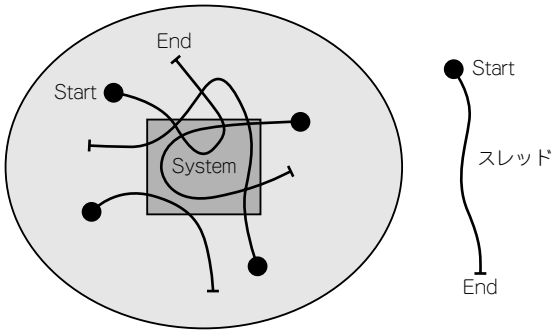
1　アクティブ・オブジェクトの方法論

アクティブ・オブジェクトの考え方は以前からあったが，昔は並列化オブジェクト指向言語と呼ばれる言語仕様の枠の中で議論されていた(1)．今のようにモデリングが重要視される前の話である．

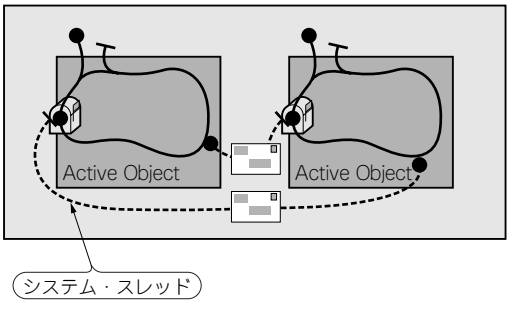
今は，プログラミング言語よりもアプリケーションに近い「モデリング・レベル」でソフトウェアを考える時代になり，特殊な言語を考えずと，CやC++，Javaでアクティブ・オブジェクトを使えるようになった．プログラミング言語に手を加えるのではなく，動くためのメカニズムとしてのフレームワークを既存の言語で記述して，その上にアプリケーション・モデルを構築するスタイルである．このスタイルでのアクティブ・オブジェクトは，プラットホームからアプリケーション・モデルへスレッド制御が上がってくるための窓口のようなものである．そして，スレッドは基本的にはアクティブ・オブジェクトに閉じ込められてアプリケーション・モデル内を渡り歩くことはない（**図1**）．このようなアクティブ・オブジェクトを使うことで，MDA（Model Driven Architecture）やxUML（Executable UML）などの動くモデルを作ることが容易になる．

UML2.0では，動くモデルを記述するためにシーケンス図やアクティビティ図が制御構造を記述できるように拡張されている(2)．これらの拡張は手続き的側面が強く，使い方をまちがうとまったく使い回しの利かないモデルを作成できてしまう．手続き的なモデルは拡張性が乏しく，反復開発の過程で容易にスパゲッティ化することが予想される．スパゲッティ化したUMLモデルは，ビジュアルな分，ただのソース・コードよりも始末が悪い．ソース・コードであれば上から下に読めば良いという流れが存在するが，ビジュアル・モデルはどこから見れば良いのか，取り付く島もない．UML2.0の制御構造記述に完全対応したツールはまだ存在していないが，既存のMDAツールでさえ，すでにこのような問題に直面している．“動く”ことは，良い面もあるが，落とし穴でもあるのだ．

筆者の経験則でいうと，スパゲッティ状態に陥る一つの原因として，アクティブ・オブジェクトとパッシブ・オブジェクトの使い分けがうまくできていないこ



（a）アクティブ・オブジェクトなしのシステム



（b）アクティブ・オブジェクトを使ったシステム

図1　アクティブ・オブジェクトはスレッドの閉じ込め

見本

とがあげられる．アプリケーション・モデル・レベルでパッシブ・オブジェクトがタスクやスレッドの制御を直接行うと，スレッドが渡り歩くモデルになってしまうため，排他制御の問題やプラットホームとの密結合の問題を引き起こす．その一方で，パッシブ・オブジェクトで十分なところにアクティブ・オブジェクトを導入して失敗することもある．たとえばモデルとしてきれいであっても，アクティブ・オブジェクトを使いすぎてリソースを浪費したため，ターゲット環境で動かすことが非現実的になってしまった場合である．

しかし，これらのことは表面的なことであり，もっと本質的な問題は「アプリケーションの並列性などの動的構造を分析段階から抽出し，それを設計に引き渡す方法論がほとんど存在していない」ことである．アクティブ・オブジェクトにはアクティブ・オブジェクト用の方法論が必要である．また，同期メカニズムなど設計の段階で，検証すべきことを実施しないで実装に入ってしまうことも問題である．たとえば，Safety（いつまで経っても悪いことは起こらない）とLiveness（いつかそのうち良いことが起こる）のような性質は設計段階で解決すべき問題であり，デバッグ段階でソース・コードを直しながらトライ＆エラーで解決できるものではない．従来のオブジェクト指向の方法論では，この辺のことがほとんど考慮されておらず，出たとこ勝負のCMMレベル1の世界になっているのが現状である．

2 動的構造の分析——服を着る順序の例

ものには順序がある．実行する順番を変えると別の結果になる場合と，順番を変えても同じ結果になる場合がある．順番を変えても結果が同一の処理は，並行に実行することができる．並行に実行することのできる処理の数が，設計工程においてタスク数やアクティブ・オブジェクト数の目安になる．したがって，開発対象ドメインに含まれる順番の制約関係を抽出するこ

Column 1　アクティブ・オブジェクトとは何か

アクティブ・オブジェクトについてはいろいろな定義や解釈があると思うが，ここでは「それ独自のスレッド制御を持っているオブジェクト」の意味で使う．UMLの定義もこのような感じである．Javaの`Thread`クラスのサブクラスや`Runnable`インターフェースを実装したクラス，MDAを実現している各種CASEツールの動的なクラスなどがアクティブ・オブジェクトと呼ばれる．RTOSのタスクやスレッド自身も，カプセル化などのオブジェクトとしての性質を満たした使い方をすれば，上記の定義に包含されるのでアクティブ・オブジェクトである．

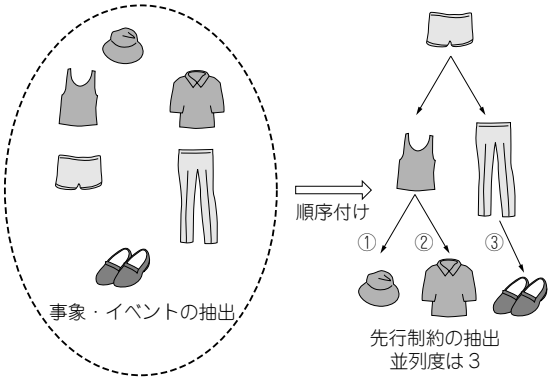
UMLではメタモデル・クラスの`Class`が`Class::isActive`という属性を持っていて，この属性が`true`ならアクティブ・クラスで`false`がパッシブ・クラスになる，という2通りしかない．したがって，スレッドと一対一で対応する形でアクティブ・オブジェクトということばを使ってしまうと，一つのスレッドに複数のアクティブ・オブジェクトを載せられるフレームワークを実現しているMDAツールで使うことばがなくなってしまう．この場合のMDAツールのアクティブ・オブジェクトをリアクティブ・オブジェクトと呼ぶこともある．

ただし，MDAツールにもいろいろあって，リアクティブ・オブジェクトをサポートしていない自称MDAツールもあるので話がややこしくなる．やはり，スレッドはスレッドと呼んでおいたほうが良い．

UMLでは，アクティブ・クラスのステレオ･タイプとして`<<process>>`と`<<thread>>`があるので，上の意味でのリアクティブ・オブジェクトをサポートしていないMDAツールは，`<<process>>`か`<<thread>>`を使用するべきだと思う．「我々は慎重にことばを選んで定義している」とUML作成に参加している某メンバがいっていたが，どんな意味にも取れるように慎重にことばを選んでいるという意見もある．

それでは，ここで扱うアクティブ・オブジェクトとは何かといえば，アクティブ・オブジェクトとリアクティブ・オブジェクト両方を対象にしている．つまり，`<<process>>`と`<<thread>>`でないアクティブ・オブジェクトを対象としている．システム全体の視点から見ると，スレッドとの関係が気になるので定義を厳密にしてほしくなるが，アプリケーション・モデルを作る立場では，スレッドもリアクティブ・オブジェクトもとくに区別する必要はない．アクティブ・オブジェクトの制御は下のフレームワークから湧き出してくるようなイメージなので，アプリケーション・モデル内ではほかのオブジェクトから呼び出されなくとも自律的に動くオブジェクトがアクティブ・オブジェクトだと理解すれば問題はない．

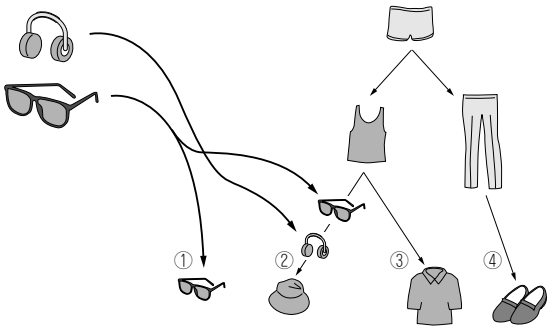
見本



(**a**) 順序付けをしていない場合　　(**b**) 順序付けをした場合

順序を導入して順序関係のつかないものどうしが並列なもの．
たとえば と

図2　洋服を着る場合の先行制約



■ヘッドフォンを追加するなら帽子の前
■サングラスは，ヘッドフォンの有無で追加する場所が変わる．ヘッドフォンがない場合，サングラスを追加することでシステム全体の並列度が4になる

図3　ヘッドフォンとサングラスを追加する場合

とが必要である．分析段階で，対象ドメインにどのようなオブジェクトがあるか，どのような事象があるか，どのようなイベントがあるかを抽出するだけでなく，これらの間の前後関係も抽出することが重要である．分析段階における動的構造とは，ここではドメインに本質的な並列度や順序制約に関する情報のことをいっている．

● 服を着る場合の分析

たとえば，洋服を着る順番を**図2**に示す．一般的なオブジェクト指向分析では，**図2**(**a**)の事象・イベントの抽出の後は，これらの抽出したものの論理的な関係や実現される機能の獲得に進んでしまい，**図2**(**b**)の事象間の前後関係にはあまり注目しない．動的構造は，前後関係を使って(**b**)のようにまとめると見えてくる．(**a**)はただの集合だが，(**b**)は順序集合になっている．順序集合は要素の順序関係を使ってハッセ(Hasse)図という図で表現できる．これが**図2**(**b**)[注1]である．図の意味は次のようになる．

「まずパンツをはく．次はズボンをはくか，ランニングを着る．ズボンとランニングのどちらを

注1：数学の本ではハッセ図は下から上に描くのが一般的である．ここでは，後でシーケンス図との対応をとるので上下が逆になっている．つまり，上から下に事象が実行される．

Column 2　プロセス代数の種類

一般にプロセス代数と呼ばれるものは，

- CSP (Communicating Sequential Processes，1978 Hoare)
- CCS (Calculus of Communicating Systems，1980 Milner)
- ACP (Algebra of Communicating Processes，1984 Bergstra)
- ATP (Algebraic Theory of Processes，1988 Hennessy)

の4種類である．表記法はそれぞれ異なる．意味付けは，どれも始めのほうは同じようだが，内部プロセスなどが出てくるあたりから扱い方が違ってくる．雰囲気的にはACPとATPは代数らしい体系である．CSPとCCSは通信系とか分散系を対象として，1980年代から実用化されるようになり，Occamやトランスピュータ，Lotos，SDLなどの成果があるが，特定の分野以外では最近は忘れられた状態に近いかもしれない．

しかし，1990年代後半から，代入文などの言語的部分を取り除いた第2次CSP，あるいはCCSをベースとしたACSR (Algebra of Communicating Shared Resources)に関する書籍が出版されるようになった．直感的には，互いに通信しながら並行動作をするもので構成されるシステムを記述する際には，プロセス代数が向いている．この互いに通信しながら並行動作を行うものがアクティブ・オブジェクトと相性が良いので，MDAへの応用が期待される．

ここではプロセス代数の入り口あたりの成果を使って，ステート・マシンの生成を行う．

見本

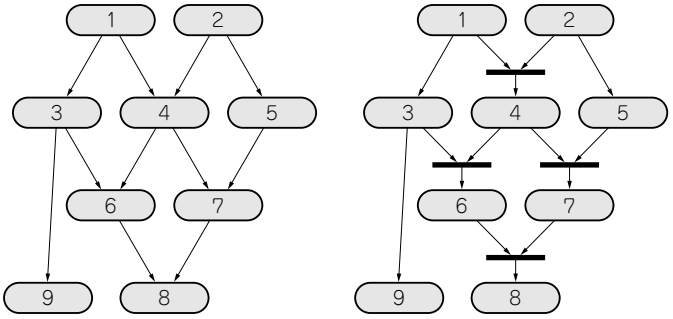


図4
アクティビティ図の例

（**a**）同期情報なし　　（**b**）同期情報付き

先にするかは自由である」…

対象ドメインに含まれる順序関係を抽出すると，ズボンとランニングのように順序関係のつかないものが出てくる．この順序関係のつかない事象や処理が並列に実行できるものになる．並列関係は順序関係の否定のような形で出てくると考えるのである．この並列関係は，同時に実行しなければならないというような強い意味ではなく，どんな順番で実行しても良いという意味である．

順序関係については，「パンツが先でズボンが後」，「ズボンが先で靴が後」，であれば「パンツが先で靴が後」というぐあいに推移関係がある．つまり，A→BかつB→CであればA→Cが成立する．しかし，並列関係では推移関係が一般には成り立たない．たとえば，自転車に乗りながら歌を歌える，歌を歌いながら自動車を運転できる，でも，自転車に乗りながら自動車の運転はできないのである．つまり，順序関係にはハッセ図で表現できるような数学的な構造がある．そのため，並列関係自身よりも数学的扱いが容易なので，順序関係を使って対象ドメインに含まれる動的構造を抽出する．

一度順序構造を抽出したら仕様の追加や変更などの際には，**図2**のようにハッセ図をメンテナンスしていく．たとえば，ヘッドフォンを仕様に追加した場合，ハッセ図のどこに入れるべきかを考える．ハッセ図への挿入位置がわかれば，システム全体の並列性への影響を評価することができる．このように，ハッセ図はインクリメンタルに再利用できる（使用の追加ごとに要素を増やしていく）ので，非常に有用である．UMLでは，アクティビティ図を使って表現できる．たとえば，**図3**のアクティビティ図は**図4**のようになる．

順序構造は，一般的に**図5**に示したところから抽出される．大きく分けると，「実世界の物理的制約」と「何をしたいか」というシステム仕様の二つから順序を抽出する．この二つが混乱すると何をやっているのかがわからなくなる．事象の順序はなるべく細かく抽出して，それらが並列で動くことを仮定してシステム環境を分析する．実際は，システムに関わる事象全体が一つのハッセ図になることはまれで，複数のハッセ図になったり，あるいは条件によって起こらない事象

Column 3　ハッセ図

集合の要素間に順序を入れた集合を順序集合と呼ぶ．その順序集合を見やすく表現するために使われるのが，ハッセ図である．

順序集合を視覚的に表現するには矢印を使って要素間の順序を表す．つまり要素が点，関係が矢印の有向グラフである．しかしそうすると，すぐに矢印だらけの図になるので，自分に対する矢印（反射的関係：ここで使用する先行制約では使わない）と推移的に導くことができる矢印を省く．また，矢印の向きを上向きか下向きに決めてしまうことで，矢印の矢を取り除いてただの線分にしてしまうとすっきりとした図を描くことができる．それが，ハッセ図である．ハッセ図や順序集合の性質の詳細については，数学の集合論の適当な入門書を参考にすると良い．二項関係とか半順序，全順序，鎖，反鎖の概念が重要である．数学の本では抽象的すぎて，つらいと思う人には以下の本をすすめる．

- 守屋悦朗，「コンピュータサイエンスのための離散数学」，サイエンス社，ISBN 4-7819-0643-5.
- C. L. Liu，「コンピュータサイエンスのための離散数学入門」，オーム社，ISBN 4-274-13007-X.

見本

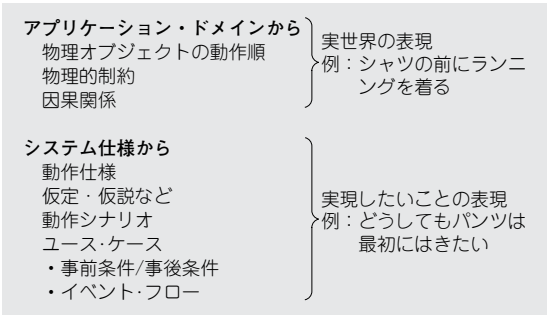


図5　分析される順序の出所

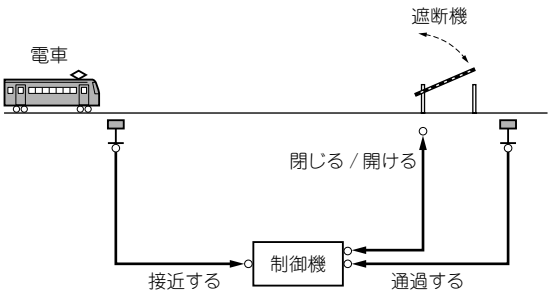


図6　踏切の例

などがあるので，部分的にステート·マシンで表現することになる．複数のハッセ図になる例として踏切制御を次に取り上げる．

3 動的構造の分析 ——踏切制御の例

● 踏切制御の分析

電車の踏切の例を**図6**に示す．踏切の仕様としては，「電車が接近したら，遮断機を閉じて，電車が通過したら，遮断機を上げる」と記述できるので，これをシーケンス図で表現して，さらに容易に制御機の動作を記述するステート·マシンまで作成することができる（**図7**）．MDA環境を使用していれば実行形式まで作ることができる．

しかし，実世界の事象は並列して起こること，外部オブジェクトは独立して勝手に動くことなどを考慮すると，これでは分析が不十分である．**図7**のシーケンス図に含まれる順序制約を，順序対に分解して，各順序対の帰属を行うと，どこが不十分であるかが明らかになる．

実際に順序対を抽出すると**図8**に示すA～Eの5種類が抽出される．そして，各順序対の帰属を行うと**図9**のようになる．ここで，P1とP2は物理現象であり制御の対象外である．Sp1とSp2が基本的なセンサと

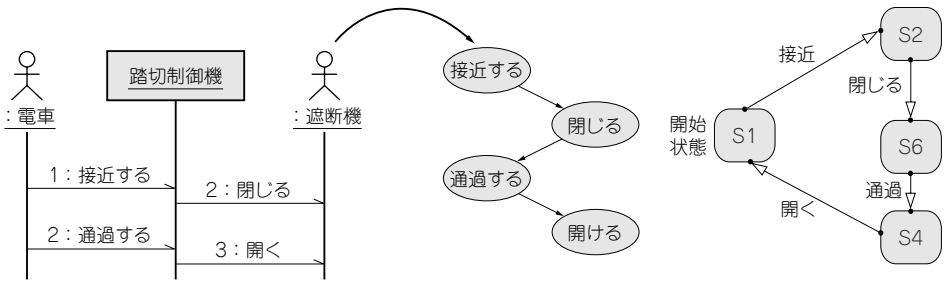


図7
一般的な手法

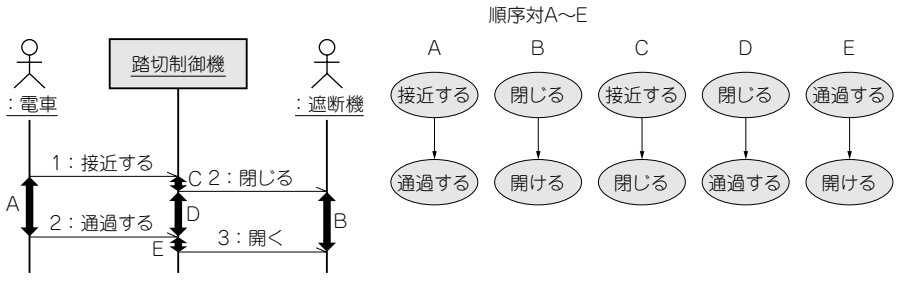


図8
順序対の抽出

注2：**図9**の説明で「自動車を止める」と補足説明をしたが，正確に自動車について言及するためにはドメイン・オブジェクトとしての自動車に関する事象も加えて議論する必要がある．ここでは簡単にするため自動車オブジェクトは省いてある．

見本