2004－SE－146　(6)
2004／11／26

アスペクト指向設計モデルの一提案

野田夏子[†]　岸　知二[††]

アスペクト指向ソフトウェア開発 (AOSD) は、様々な関心事 (concern) をモジュール化する新しいメカニズムとして、ソフトウェア開発の様々なフェーズにおいてアスペクトを利用するものである。我々は、アスペクト指向技術を設計レベルに適用することを検討している。本稿では、設計上の関心事をアスペクトとして独立に設計し、それらのアスペクトが連動してひとつのシステムの振る舞いを表現する、アスペクト指向設計モデルを提案する。本設計モデルでは、アスペクト間の関係を規則として与えることにより、それぞれのアスペクトを独立性高く記述できるとともに、複数のアスペクトから成るシステム全体の振る舞いを設計することができる。

A Proposal of Aspect-Oriented Design Model

Natsuko Noda[†] and Tomoji Kishi[††]

Aspect-oriented software development (AOSD) is now getting to be applied, in which we use aspect as a new mechanism to modularize various concerns. We are examining to apply aspect-orientation to software design. In this paper, we introduce an aspect-oriented design model, by which we describe the target system in terms of separated aspects. We also define the relationships among aspects by means of rules, that makes each aspect highly independent from other apsects, and also makes it possible to "weave" aspects into one thereby we can design whole system behavior.

1. はじめに

近年、アスペクト指向ソフトウェア開発への関心が高まっている[3)]。アスペクト指向ソフトウェア開発は、従来の単一視点からの構造化では局所化できなかった横断的な関心事 (concern) に関し、複数視点からシステムを構造化して適切な局所化を行うとするものである。

アスペクト指向技術は、まずプログラミングレベルの技術として誕生し、発展してきた。しかし、そもそもシステムをいかに適切に構造化するかという問題は、設計段階における重要な課題である。そのため、最近は設計レベルにおいてもアスペクト指向技術が有効に適用できると期待され、研究がされ始めている。

このような研究では、特定のアスペクト指向プログラミング言語の利用を想定して、言語上に現われる概念をどのように設計するか、あるいはそれを UML でどのように表現するか、といったことが多く検討されている。しかし、設計段階におけるアスペクト指向技術の適用においては、そうした問題だけでなく、プログラミング言語には依存せず、そもそもどのような関心事が設計段階においてアスペクトとして捉えられるべきか、またそれらのアスペクト群をどのように記述するか、そしてそれらのアスペクト群をどのようにひとつのシステムの設計へと構成するのか、といった問題も検討されなければならない。

我々は、アスペクト指向技術の設計への適用を検討している[6)]。そこではアスペクトをどのようにモデル化し、そのモデルがどのような実行意味を持つか等、アスペクト指向設計モデルを明らかにすることが、重要な課題のひとつである。本稿では、アスペクト指向設計を実現するための設計モデルの一提案をする。2 章で提案する設計モデルを説明し、3 章で例題を用いて具体的にモデルを解説する。4 章では、関連研究との比較等を交え、本設計モデルについて考察を行う。

2. アスペクト指向設計モデル

我々の提案するアスペクト指向設計を、従来の設計と対比させて図 1 に示す。

従来の設計では設計対象は単一の視点からモデル化されるが、アスペクト指向設計では複数の関心事に対応した複数の視点からモデル化される。ここで特定の

† NEC
NEC Corporation
†† JAIST
Japan Advanced Institute of Science and Technology

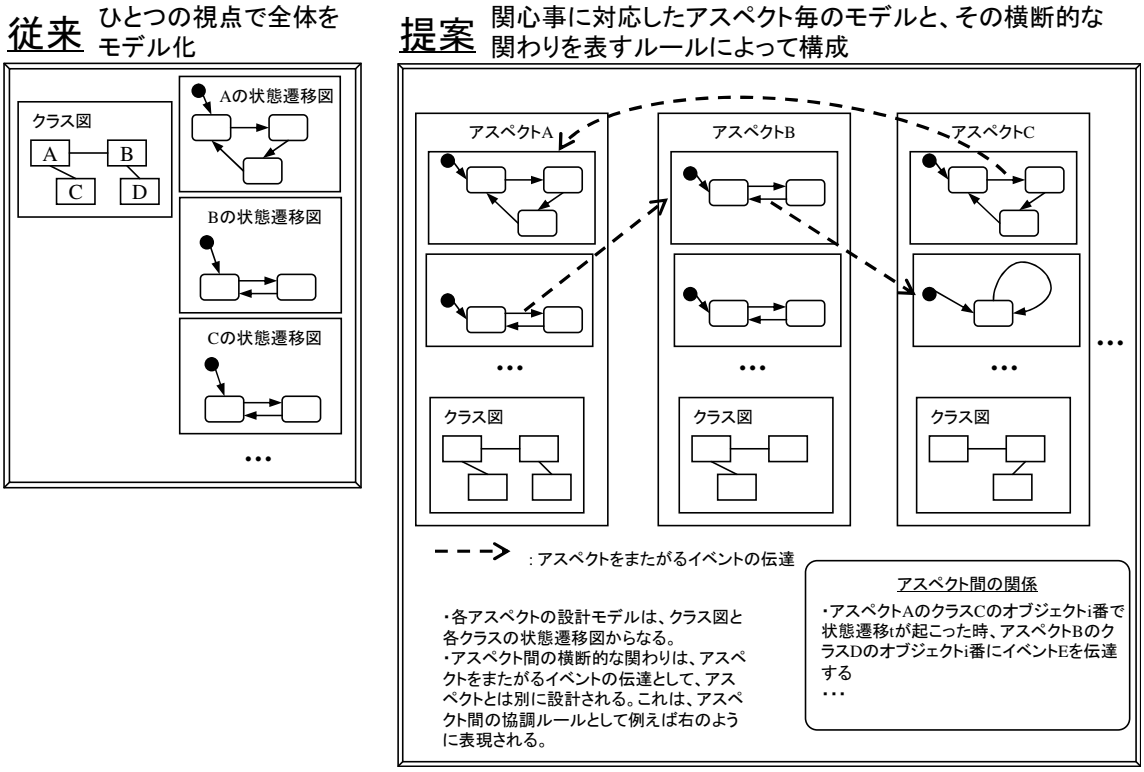


図 1 アスペクト指向設計全体像

関心事に基づいて検討される (その関心事を実現するための) ソフトウェア構造をアスペクトと呼ぶ。特定の関心事の表現について、必要なことは対応するアスペクトの中に定義されており、他のアスペクトに依存しないものとする。すなわち、各アスペクトは、他のアスペクトに依存せず、独立して設計することができる。このようなアスペクトとしては、典型的には以下のようなものが考えられる：

- 特定の役割だけに注目した協調動作
- 論理的振る舞いを実現するためのメカニズム
- 特定の状況における処理 (エラー処理等)

アスペクトは関心事の視点からの構造の射影であるため、複数のアスペクトに同一の設計対象がそれぞれの視点で出現しうる。従って例えばひとつのアスペクトにおいて行った設計対象に対する操作が、他のアスペクトに出現する同一設計対象のモデルに影響を与えるなど、アスペクトは相互に関連を持つ。そのため、複数のアスペクト群によってひとつのシステムの設計を構成する際には、あるアスペクトにおける何らかの動作の後に別アスペクトで特定の動作をするなど、アスペクト間の相互関連を定義する必要がある。

このような設計を実現するための、アスペクト指向設計モデルを提案する。

各アスペクトの設計モデルは、通常のオブジェクト指向設計モデルに準ずるものとし、静的な構造をクラス図で、振る舞いを各クラス毎の状態遷移図で与える。ただし、本稿では簡単のために、状態遷移について階層化した状態は除くものとする。

ここで、設計モデルの実行意味は以下のように与える。まず、各アスペクト内の各オブジェクトは、独立に状態遷移を行う。各オブジェクトは入力イベントのキューを持ち、FIFO で順にイベントが発火し状態遷移が引き起こされる。なお各アスペクトは、対応する関心事に関わることのみを定義するのであるから、遷移が起こった時の動作として他アスペクトに影響を及ぼす動作は行わない。

一方前述したアスペクト間の相互関連の定義として、イベントを介したアスペクト間の協調動作のルールを与える。

アスペクト間の協調動作は、一般にはあるアスペクトでの状態変化等の事象が、他のアスペクトでの状態変化のトリガになって引き起こされると考えられる。そこで、協調動作のルールでは、どのアスペクトでどんな状態遷移が起こると、別のどのアスペクトにどう

いうイベントとして伝達するか、ということを記述する。ここで伝達されるイベントは、そのイベントが伝達される側のアスペクトで解釈できるイベントでなければならない。また、イベントを伝達する際には、オブジェクトを特定する必要があるので、オブジェクトには index 番号を持たせるものとする。index 番号は、異なるアスペクト内にあって対応するオブジェクトには同じ番号を持たせるようにし、動的にオブジェクトを生成する場合にも、index 番号を与えて生成することにする。

したがって、協調動作のルールは、それぞれが以下のどれかの形式を持つ複数のルール群とする。

- アスペクト A のクラス C のオブジェクト i 番で遷移 t が起こった時、アスペクト A' のクラス C' のオブジェクト i 番にイベント E を伝達する
- アスペクト A のクラス C のオブジェクト i 番で遷移 t が起こった時、アスペクト A' のクラス C' の全てのオブジェクトにイベント E を伝達する
- アスペクト A のクラス C のオブジェクト i 番で遷移 t が起こった時、アスペクト A' のクラス C' の唯一のオブジェクトにイベント E を伝達する

なおルールの記法については本稿の提案の範囲外である。

このように本実行意味では、複数のアスペクト群が、与えられた関係 (ルール) に基づいて協調動作をする形で、全体としてひとつのふるまいを示すようになっている。アスペクトをマージ (アスペクト毎の複数のクラス図から全体を示すひとつのクラス図に合成したり、合成されたクラス単位にひとつの状態遷移図を複数の状態遷移図から合成する等) した結果に対して実行意味を定義するものではない。

このことは本設計モデルの適用に自由度を与えている。例えば設計段階で本モデルを利用してアスペクトとその間の関係を設計、実行確認した上で、アスペクトをマージしてひとつの設計モデルとし、実装段階へと移行してもよいし、あるいは設計段階ではアスペクトは分離したままとして、実装段階でそれらをアスペクト指向言語を用いて実装するといった利用も可能となる。

3. 例　　題

本章では、簡単な例題を用いて提案する設計について説明する。

本例題は、画面アップデートに関する例題で、数をカウントしているカウンタクラスが持つ数字を、各ユーザ毎に対応づけられているディスプレイに表示させるものである。ディスプレイは、カウンタの数が変更される度に表示を更新するとともに、カウンタの持つ数がユーザ設定の閾値を超えると点滅表示する。ここでは、内容の変更により表示が更新される画面アップデートの機能を、オブザーバーパターンの持つ協調動作のメカニズムを用いて実現する。なおデザインパターンを用いた設計や実装にアスペクト指向を用いることはいろいろ試みられており[1)2)]、特にオブザーバーパターンのアスペクト指向言語での実装は、典型的な例題として AspectJ[4)] や Hyper/J[5)] の解説にも用いられている。

この例題には、以下の 3 つの関心事が含まれる：

(1)　数をカウントする機能
(2)　データを表示する機能
(3)　データをビューに反映させる機能

これらの関心事を実現するソフトウェア構造を、独立したアスペクトとして設計する。すなわち、数をカウントする機能を Counter アスペクト、データを表示する機能を Display アスペクト、データをビューに反映させる機能を Collaboration アスペクトとし、それぞれのアスペクトについて静的な構造をクラス図を用いて、またそれぞれのクラスの動的な振る舞いを状態遷移図を用いて設計する。それぞれのアスペクトの設計モデルを図 2(a) に示す。また、この 3 つのアスペクト群のそれぞれの間の関係を与えるルール (2 章参照) を、図 2(b) に示す。

このルールにしたがって、例えば次のような一連の振る舞いが実行される;「Counter オブジェクトで数が更新されると (遷移 t1)、イベント change が Subject オブジェクトに伝達されることにより、Subject オブジェクトで遷移 t1 が実行される。この遷移により、Subject オブジェクトから Observer オブジェクトにイベント notify が伝達される。さらに Subject オブジェクトにイベント getState が伝達され遷移 t2 が実行される。この遷移により Observer オブジェクトにはイベント reply が伝達される。そしてこのイベントにより Observer オブジェクトがアップデートされると (遷移 t2)、Display オブジェクトにイベント update が伝達され、Display オブジェクトのアップデートが実行される。」

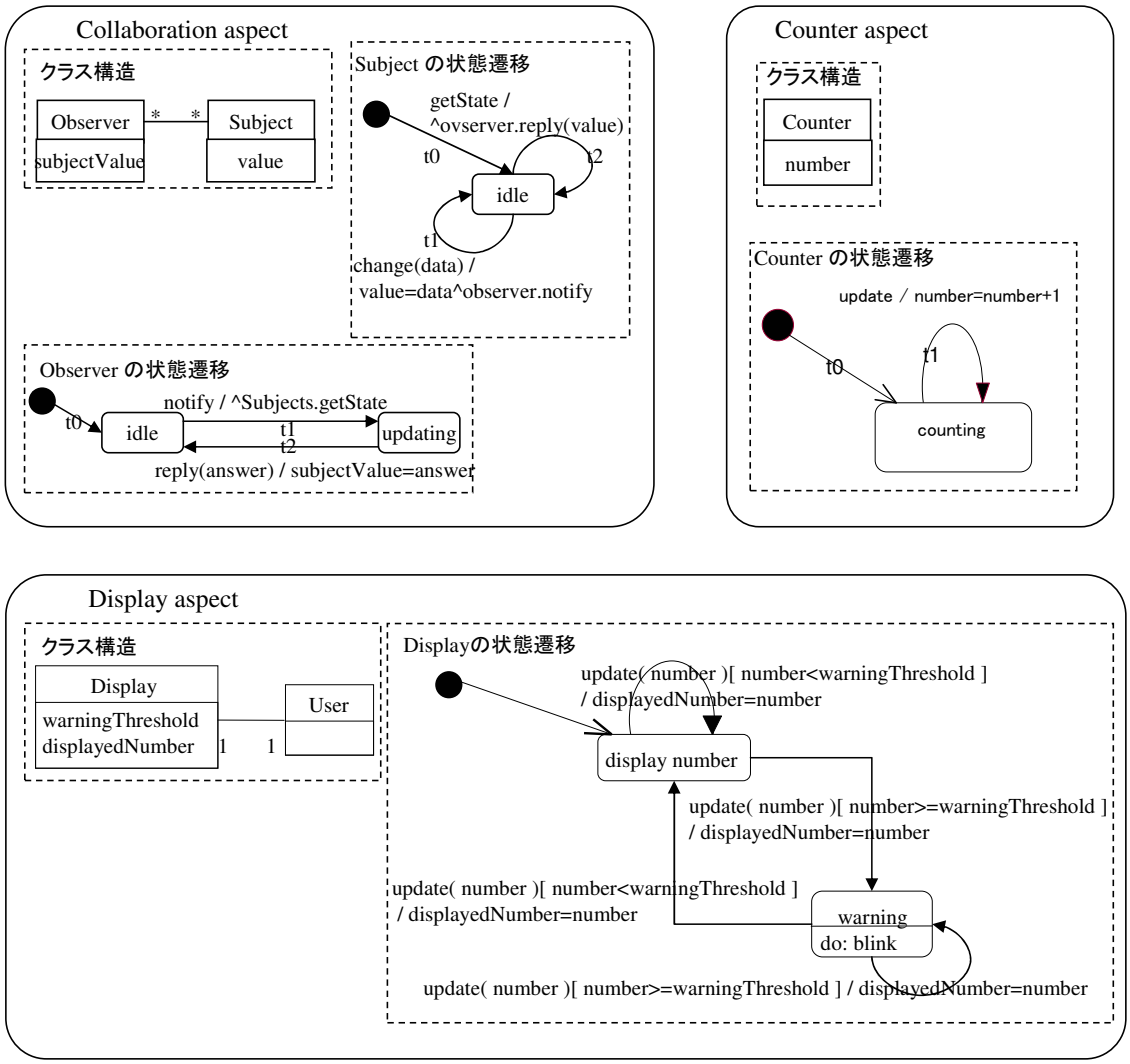
4. 考　　察

本章では、本稿で提案した設計モデルについて考察を行う。

4.1 アスペクトの定義方法

現在提案されている様々なアスペクト指向プログラ

(a)



(b)

連動ルール
- 初期オブジェクト subject(1), observer(1), observer(2), counter(1), display(1), display(2)
- アスペクトCounterクラスCounterのオブジェクトi番で遷移t1が起こった時、
  アスペクトCollaborationクラスSubjectのオブジェクトi番にイベントchange(Counter.number) を伝達する
- アスペクトCollaborationクラスObserverのオブジェクトi番で遷移t2が起こった時、
  アスペクトDisplayクラスDisplayのオブジェクトi番にイベントupdate(observer(i).observeeValue)を伝達する

図 2 例題の 3 つのアスペクトとアスペクト間の関係

ミング言語や、アスペクト指向での設計の研究において、アスペクトとは単一の階層構造では捉えきれない横断的な関心事を扱うための仕組みであるということは、共通的な認識であると思われる。しかし、それらの複数の関心事の中で、本質的に必要とされる関心事、あるいは全体の下敷きとなる関心事 (core concern とも呼ばれる) が必要であるかどうかについては、意見が分かれるところである。

本設計モデルは core concern を必要としないモデルである。アスペクトは関心事毎に独立して設計されるものであり、下敷きとなるクラス構造に関心事に従って追加・修正を行うものがアスペクトとなるわけではな

い。各アスペクトはそれぞれが対象となるソフトウェア構造のある視点からの射影であって、その意味でそれぞれは対等であり、どれかが主でその他が従ということではない。この考え方は、Hyper/J の基礎となっている hyperspace モデルに類似している。すなわち hyperspace モデルでは、特定の関心事を hyperslice として他の hyperslice に依存することなく独立して定義する。

core concern を必要とするモデルと必要としないモデルのどちらが良いかは、それぞれに特徴があり、簡単に断じることができないものではあるが、core concern を想定したモデル化では、他のアスペクトの定義は core concern に依存したものとなり、アスペクトを独立性高く設計することが難しくなる。本設計モデルでは、各アスペクトは他に依存することなく設計できるので、独立性が高くなり再利用の可能性も広がる。より自由度の高い設計を実現できると考えられる。

4.2 アスペクト間の相互関連

本設計モデルでは、各アスペクトを全く独立に定義するので、全体の振る舞いを設計するためにはアスペクト間の相互関連をアスペクト自体とは別に与えなければならない。

このアスペクト間の相互関連の定義方法については、hyperspace とは別のアプローチを取った。hyperspace モデルでは、hyperslice の構成要素 (クラス、属性、メソッド等) について、同一である、一方で他方を書き換える等の構成要素間の関係を与えることにより、全体でひとつのシステムを表現する。これに対して本提案では、各アスペクト間でイベントが伝達されることにより全体の協調動作を実現する。

これは、むしろ AspectJ での Join point モデルに類似している。Join point とはプログラム実行上で定義される時点 (メソッドが呼ばれた時点、属性値を読み出した時点等) を指すが、Join point モデルでは、join point に対して、その時点で何をするかを advice として定義することにより、相互関連を定義する。提案するアスペクト指向設計でのアスペクト間の関係ルールにおける、「アスペクト A のクラス C のオブジェクト i 番において状態遷移 t が起こった時」というのは join point に相当し、その時点で「アスペクト A' のクラス C' のオブジェクト i 番にイベント E を伝達する」というのが advice に相当すると考えることができる。もちろん、AspectJ はプログラミングレベルの記述であり、我々の考える設計レベルの記述とは記述の構成要素に違いがあるが、アスペクト間の関連付けのモデルとしては類似性を見ることができる。

本設計モデルでは、異なるアスペクト間でのオブジェクトや属性の同一性を定義して関連付けることはしていない。ソフトウェア構造を眺める視点が異なれば、その視点におけるモジュール化単位はそれぞれ異なりうるので、同一性を議論することが難しいからである。しかし、hyperspace モデルの基になっている Subject-oriented-programing では、各 Subject を実体の ID と対応付け、異なる Subject 間の同一性を議論するアプローチを取っている。各アスペクトがあるソフトウェア構造、つまり実体の様々な視点からの射影であるとすれば、本来的にはどこかに実体は存在するものであり、実体との対応付けは可能であろう。しかし、何が実体であるのかは、また様々な捉え方がある。そのため、本設計モデルでは、実体との対応付けをするのではなく、Join point モデル的に各アスペクトの連動を与えるアプローチを取った。両者の本質的な特徴、実開発における利用のしやすさ等、メリットデメリットを今後比較したい。

4.3 複数アスペクトの実行の制御

前節で述べたように、本設計モデルでは各アスペクト間でイベントが伝達されることにより、全体の協調動作が実現されるが、このようなアスペクトの連動のさせ方については、本提案以外の方法も考えられる。例えば状態への入退場時や入退場アクションの前後といったさらに詳細な時点でイベントの伝達を行うモデルにすれば、より詳細な制御を行うこともできるであろう。どういう制御が設計上どのような意味を持つのか等については、今後さらに検討が必要である。その上で、どこまでの制御が設計レベルで必要とされるのか検討し、必要なものは設計モデルに取り込んで行きたい。

また、各アスペクトの実行の順序に関し、本設計モデルのアスペクト間の関係の与え方では、各アスペクトの実行の半順序は保証されるが、全順序を規定することはできない。全順序の規定が設計上どういう局面で必要となるかについても検討が必要である。どのレベルの設計において、またどんな問題を対象とする場合に全順序を規定したいのかを検討し、必要であれば全順序が規定できるように本設計モデルを拡張することを考えたい。

4.4 設計レベルにおけるアスペクト指向適用の意義

一般に、アスペクト指向の適用は、単一視点で全体を構造化する場合と比較して、拡張や修正に有利であると期待される。しかしながら安易にアスペクトを導入し、それらの間をアドホックに関連付けていては、全体の構造は決して望ましい構造とはなりえない。従っ

て何をアスペクトとし、それらをどのように相互関連付けするかについて、設計段階で方針づけることが重要となる。本設計モデルのねらいのひとつは、アスペクト指向言語よりも抽象度の高いレベルでアスペクトやその間の関連を設計可能とすることにより、こうした問題を改善することにある。

今後実問題の記述評価などを踏まえ、妥当な設計表現や言語メカニズムとの対応などについて検討が必要であろう。

5. おわりに

本稿では、アスペクト指向設計を実現するための設計モデルの一提案を行った。本提案のモデルでは、アスペクト間の関係を規則として与えることにより、それぞれのアスペクトを独立性高く記述できるとともに、複数のアスペクトから成るシステム全体の振る舞いを設計することができる。今後は多くの具体的な例題に適用しながらモデルの完成度を高めて行きたい。

参考文献

1) Hannemann, J., and Kiczales, G.: Design Pattern Implementation in Java and AspectJ, OOPSLA2002, 2002
2) Noda, N., and Kishi, T.: Design Pattern Concerns for Software Evolution, IWPSE2001, 2001
3) http://aosd.net
4) http://www.eclipse.org/aspectj/
5) http://www.research.ibm.com/hyperspace/ HyperJ/HyperJ.htm
6) 岸知二, 野田夏子: アスペクト指向による状況モデリング, 情報処理学会組込みソフトウェアシンポジウム ESS03, 2003