アスペクト指向モデリングメカニズムの動的意味について

野田夏子[†1]　杉田郁人[†1]　岸知二[†2]

概要：我々は過去に，ソフトウェアの柔軟性を高めるためのメカニズムとして，アスペクト指向モデリングメカニズムの提案を行った．本メカニズムでは，関心事をモジュール化するアスペクトとそれらアスペクト間の関係をルールとしてアスペクトとは独立に定義することにより，柔軟なソフトウェア構成の実現が可能である．現在我々は，本メカニズムに基づくモデル駆動開発環境を開発しているが，モデルから実行可能なコードを生成するためにはモデルの動的意味の明確化が必須である．そこで本稿では，アスペクト指向モデリングメカニズムの動的意味を提案する．

キーワード：アスペクト指向モデル，モデリングメカニズム，動的意味

Aspect-Oriented Modeling Mechanism - Dynamic Semantics -

NATSUKO NODA[†1]　IKUTO SUGITA[†1]
TOMOJI KISHI[†2]

1. はじめに

ソフトウェアは現代社会において凡そあらゆるものに組み込まれ，社会の一つの重要なインフラとなっている．そのため，ソフトウェアはますます巨大化，複雑化が進み，効率良くまた信頼性高く開発するためには，再利用が欠かせなくなっている．さらに，再利用の成功のためには，単に再利用部品が存在しても不十分で，再利用部品により全体を柔軟に構成できることが非常に重要になってきている．

我々は過去に，ソフトウェアの柔軟性を高めるためのメカニズムとして，アスペクト指向モデリングメカニズムの提案を行った[3][4]．このメカニズムでは，ソフトウェアに対する複数の関心事をアスペクトとしてモジュール化することにより，ソフトウェアを関心事に従って分割することができる．各アスペクト間の関係をアスペクト内には一切定義せず，アスペクト関連ルールによって関係づけることで，各アスペクトの独立性が高められ，ソフトウェア全体を柔軟に構成することが可能である．本メカニズムにおけるアスペクトは，AspectJ[1]等に見られる，基盤構造の上に付加的に定義されるアスペクトよりも，[6]で提案された多次元的に分離されそれぞれが独立な関係にある関心事に近い．ソフトウェアが全体として動作する際に生じる関心事間の関係の定義を，関心事そのものの定義からは切り離すことにより，より柔軟性を高めている．

本メカニズムを具体的なソフトウェア開発で利用するため，我々は最近本メカニズムに基づいて設計されたアスペクト指向モデルから実行可能なソースコードを生成するアスペクト指向モデル駆動開発環境の開発に着手した[5]．近年見られる，ソフトウェアプロダクトライン等の類似した多品種の開発の増加や，IoT システム等様々な機器との接続が必要な開いたシステムの増加に伴い，ソフトウェアの構造の柔軟化がさらに求められており，柔軟な設計モデルに基づいて具体的なソフトウェアを迅速に得ることが求められているためである．

モデルから実行可能なコードを生成するためには，モデルによって表現される動的な意味を明確にしなければならない．これまでに提案したアスペクト指向モデリングメカニズムでも，アスペクトの構成要素となるクラスがそれぞれに状態遷移を持ち通常の状態機械として動作すること，アスペクト関連ルールによって他のアスペクト内のクラスの状態遷移を引き起こすイベントを発生させられること等，基本的な動作を定義している．しかし，例えば同時に発生し得るイベントの処理順序等，より詳細な動作については規定していなかった．そこで，本稿では，既発表のメカニズムに基づくアスペクト指向モデリングメカニズムの動的意味を提案する．なお，アスペクト指向によるモデリングや設計手法は他にも複数提案されており，一般にアスペクト指向モデルという場合には我々の提案メカニズム以外のものも当然含まれるが，本稿において以降アスペクト指向モデルとは，我々の提案メカニズムによってモデル化されたものを指すものとする．

以下，2 章で我々が過去に提案したアスペクトモデリングメカニズムの概要について説明し，3 章でアスペクト指向モデリングメカニズムによるモデルの動的意味を提案する．4 章で本提案の有用性や今度の課題について議論し，5 章で本稿を締めくくる．

2. アスペクト指向モデリングメカニズム

本章では，過去に我々が提案したアスペクト指向モデリングメカニズム[3][4]について説明する．本メカニズムでは，アスペクトとアスペクト関連ルールにより，複数の関心事

†1 芝浦工業大学
Shibaura Institute of Technology
†2 早稲田大学
Waseda University

© 2018 Information Processing Society of Japan

に対応した複数の視点でソフトウェア全体をモデル化する．

アスペクトは関心事をモジュール化する単位で，対応する関心事の視点から見たソフトウェア全体の射影である．クラス図はアスペクトの静的な構造を示し，クラス図中の各クラスはその振る舞いを示すステートマシン図を持つ．アスペクト間の関係はアスペクト内には定義されずアスペクト関連ルールによって定義されるため，アスペクトは自己完結するように定義される．

アスペクト間の関係として，以下の 2 種類を定義する．

- あるアスペクト内のクラスの状態遷移が他のアスペクト内のクラスの遷移を引き起こすイベントとなって，アスペクトの振る舞いが関係づけられる．
- あるアスペクト内のクラスの状態が，他のアスペクト内のクラスにおける状態遷移のガード条件となって，アスペクトの振る舞いが関係づけられる．

アスペクト関連ルールは，これらの関係を，ステートマシン図における，遷移に関するイベント，遷移，ガード条件を用いて独自の構文を用いて記述するもので，以下の 2 種類がある．図 1 に示すアスペクト指向モモデルの例を用いながら，それぞれのルールの記法と意味を説明する．

- イベント導入ルール : 状態遷移が発生したときに特定の動作をトリガするために他のアスペクトにイベントを導入する．例えば，”SAFECTRL.SafeContol:t2->off ^SERIVCE.Service” は，遷移 t2 がアスペクト SAFECTRL のクラス SafeControl で発火したとき，イベント off がアスペクト SERVICE のクラス Service に導入されることを意味する．
- 条件参照ルール : 特定のクラスの状態遷移が発火するときに他のアスペクトのクラスの状態を参照し，ガード条件とする．例えば，”SERVICE.Service:t1[!(SAFECTRL.SafeControl@Alarm)]”は，アスペクト SERVICE のクラス Service の遷移 t1 は，アスペクト SAFECTRL のクラス SafeControl の状態が Alarm でないときではないと発火できないことを意味する．

本メカニズムのルール記述においては，クラスと遷移の指定にワイルドカードを使用することができる．

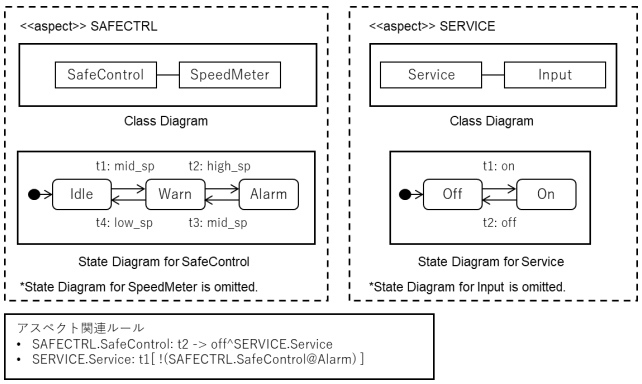

図 1 アスペクト指向モデリングメカニズムによるモデル例

例えば，”A1.*:t1->E1^A2.C2”は，遷移 t1 がアスペクト A1 の任意のクラスで発火したとき，イベント E1 はアスペクト A2 のクラス C2 に導入されることを意味する．

3. 動的意味

[3]および[4]で提案したメカニズムにおいては，モデルのモジュラリティの観点に特に注目してメカニズムが説明され，それを用いて設計される構造の柔軟性について考察されてきた．本稿では，組込みソフトウェアなど振る舞いの設計がより重要になる分野への応用や，コード生成の実現などのために，アスペクト指向モデルの動的意味をより明確にする．基本的な方針としては，従来の UML で規定されている動的意味に基づき，矛盾なく利用できるものはそのまま利用し，アスペクトを導入したことにより拡張が必要な部分のみ拡張することとする．

3.1 モデル要素の意味

動的意味を提案するにあたり，まずは各モデル要素の意味について必要な明確化を行う．また提案の第一段階として無用な複雑化を排除するためにモデル要素に制約を課す．

(1) アスペクト

アスペクトは，関心事をモジュール化する単位であり，対応する視点から見たソフトウェア全体の射影である．つまり，アスペクトは特定の観点に関わるモデル要素をグループ化するものであり，パッケージングの機構と捉えることもできる．モデル化に際しては，名前空間として用いることもできる．なお，これは静的な構造におけるモジュール化の機構である．動的な構造に関して，アスペクトそれ自体を並行動作の単位とはしない．

(2) クラス，オブジェクト

アスペクトの静的な構造はクラス図で示される．つまり，アスペクトには複数クラスが含まれ得る．

1 クラスからは複数オブジェクトが生成され得る，すなわちアスペクトには複数種類のクラスそれぞれに対応する複数のオブジェクトが含まれることになるが，現段階では 1 クラスから生成されるオブジェクトは一つのみという制約を置くことにする．これは，オブジェクト構造の定義とルール記述の簡約化のためである．

クラスの継承については，まずは考慮しないこととする．これは，継承したクラスの状態遷移と親クラスの状態遷移の関係について UML では規定されておらず，複数の可能性が考えられるからである．

(3) 状態遷移

クラスに対して，ステートマシン図によって状態遷移が定義される．各オブジェクトは，クラスに定義された状態遷移に従って動作することになり，オブジェクトが一つの状態機械になる．

UML のステートマシン図の記法においては，各遷移に明示的に名前をつけることはしないが，本アスペクト指向モ

© 2018 Information Processing Society of Japan

デルの記述においては，全ての遷移に明示的にユニークな名前をつけるものとする．これは，アスペクト関連ルールの記述においてルールが関係する遷移を特定する必要があるが，それを名前によって簡単に行うためである．本モデルにおいては，便宜上，全ての遷移にそのステートマシン図内でユニークな名前をつける．

なお，状態遷移の定義において，簡単化のため，入れ子の状態は考慮しないこととする．

3.2 モデリングメカニズムの動的意味

前節で述べた制約の下でモデル化した複数アスペクトについて，個々のアスペクト(アスペクトに対応するソフトウェア全体から射影された部分構造)を個別に動作させる場合，その動的意味は UML に定義された動的意味と同様である．つまり，アスペクト内の各オブジェクトがそれぞれ並行動作を行う．オブジェクトはそれぞれがイベントキューを持ち，発生したイベントはイベントキューに入る．イベントキューからのイベントの取り出しと，取り出したイベントによる遷移の実行は，UML の状態機械に定義される run-to-completion に従う．

これらのアスペクトがアスペクト関連ルールで関連づけられた場合，アスペクトとルールの全体で表現されるモデルの振る舞いを，以下のように定義する．

- 遷移 t を対象とする条件参照ルールがあれば，該当するルールに記載された条件を，遷移 t のガード条件に加える．同じ遷移 t を対象とする条件参照ルールが複数あれば，それらすべてのルールに記載された条件と，(あれば)もともと遷移につけられていたガード条件のすべてのアンドを取ったものがガード条件となる．
- 遷移 t を対象とするイベント導入ルールがあれば，該当するルールに記載されたイベントの導入，すなわちイベントが発生したことを他アスペクト内のクラスに通知する(つまりイベントを対象となるオブジェクトのイベントキューに追加する)ことを，遷移 t のアクションに追加する．
- 同じ遷移 t を対象とする複数のイベント導入ルールがあれば，対応する複数のイベント発生の通知の順番を，ルール設計者がルール設計時に明示的に示すものとし，示された順番で導入する．この順番は，ルール全体に対して絶対順序として示す．もしも順番が明示されなければ実装依存となる．
- 条件参照ルールに記載された他のアスペクト中のクラスの状態は，即時に観察でき，その状態にあるかどうかの真偽が決定される．
- イベントはどこで発生したものであっても，イベントキューへの入出力で区別されない．すなわち，クラスの状態遷移に定義されているイベントも，イベント導入ルールにより付加されたイベントも，自アスペクト内で発生したイベントも、ルール発火によって発生したイベントも，イベントキューへの入出力においては区別なく扱われる．

上記に従って，アスペクト関連ルールを各クラスの状態遷移に展開することができ，アスペクト指向設計モデル全体は，振る舞いが状態遷移によって定義された並行動作する複数オブジェクトの集まりを表現することになる．アスペクト関連ルールが展開された状態遷移の動的意味は，UML の状態機械の動的意味と全く同じである．すなわち，個々の状態機械においてイベントの処理は，run-to-completion で処理される．つまり，イベントキューからイベントが取り出されると，そのイベントがトリガとなる遷移が一つ発火し，遷移前の状態から退場し，遷移元の状態に退場アクションがあれば実行し，遷移上のアクションを実行し，遷移先の状態の入場アクションを実行して状態に入場し，そこで安定した状態になる．この一連の処理が完了するまで，イベントキューからの次のイベントの取り出しは行われない．

4. 議論

本稿では，我々が過去に提案したアスペクト指向モデリングメカニズムに対して，その動的意味を新たに定義した．この動的意味によって設計モデルを解釈することにより，UML に基づく従来のオブジェクト指向によるモジュール化と状態遷移による各モジュールの振る舞いの定義と同等のモデルが得られる．このように，アスペクト指向モデルは従来のオブジェクト指向の枠組みに自然にマップすることができるので，既存パラダイムのモデルやコードへの変換・生成を自然な方法で実現できる可能性が高い．

一方で，従来の UML では，各状態機械の実行意味は定義しているが，複数の状態機械の振る舞いの制御に関しては何も定義していない．例えば，ある状態機械が解釈可能な複数のイベントが論理的には同時に発生した場合に，イベントキューにどのような順番でイベントが入るのかは規定されていない．また，複数のオブジェクトが並行動作する時，それぞれが全く対等に動作するのか，動作に優先順位があるのかは，UML の規定の範囲外となっている．本稿で提案した動的意味も，UML と同レベルの定義しかしていないので，こうした実行制御に関しては何も定義されていない．モデルを現実の実行環境において実行可能なコードに対応させるには，このような全体の実行意味に関しても詳細に定義する必要がある．この点は，UML と同様に semantic variation point と考えて規定の範囲外としている．このような実行意味には複数のものがあり，どれが妥当であるかは対象ドメインや実装時に利用できる環境に依存するため，モデルにおいてただ一つの実行意味を固定することは，かえってモデルの利用局面を限定することになると考えたためである．今後，オブジェクトの優先度やイベントの優先度等が定義できるプロファイルを作成し，具体的

© 2018 Information Processing Society of Japan

な利用局面に応じてプロファイルを利用したモデル化ができる方法等を検討する．

本アスペクト指向モデルは，注目する関心事の視点毎にアスペクトに分割し，一つのアスペクトのモデル化の際には他のアスペクトとの関係を全く考慮せずにそのアスペクト固有の特徴をモデル化するので，各アスペクトの理解が容易になる．また，アスペクトはどう接続されるかという想定なしに定義され，その接続方法はルールで定義するために，接続の自由度が上がり，ソフトウェアアーキテクチャをより柔軟に構成することが可能である．近年，変化に強いアーキテクチャを設計するため，例えばマイクロサービスアーキテクチャ[2]の考え方に見られるように，細粒度のコンポーネントをフレキシブルに構成するというアーキテクチャ構成方法が求められているが，本アスペクト指向モデルは，このような目的に適している．さらに，標準的な run-to-completion にマッピングできるので，本アスペクト指向モデルから既存のパラダイムでのモデルやコードへの変換に親和性があると考えられる．

本稿で提案したアスペクト指向設計モデルでは，簡単化のためにいくつかの制約を置いた．これらの制約を解消することは今後の課題であるが，以下に示すように比較的な軽微な修正で解消できると考えている．

- 1 クラスから生成されるオブジェクトは 1 つ：この制約を外すためには，実行開始時のオブジェクト構成の与え方を決定する必要があるが，これにはオブジェクト図等を利用可能である．また，現状アスペクト内のクラスを指定してアスペクト関連ルールを記述するが，オブジェクトが特定可能なようルール記法を拡張する必要がある．オブジェクトの命名規則を決めれば，それに従いルールの拡張ができると考える．
- クラスの継承を考慮しない：継承を用いるためには，子クラスの状態遷移が，親クラスの状態遷移とどのような関係にあるかを決定しなければならない．クラス階層と状態遷移に関しては既に多くの研究があるので，それらに基づいて状態遷移の継承を定義することにより，この制約は解消できる．
- 入れ子の状態は考慮しない：入れ子の状態に関しては，単純な状態名だけでは状態を特定できず，一番外側の状態からの階層まで含めて明示しなければ状態を特定できなくなるので，アスペクト関連ルールの記述を拡張しなければならない．階層を含めた状態名を記述する，あるいは状態名に関する名前付けの規定を作ることにより，この制約は解消できる．ただし，状態を階層化するということは，抽象的な振る舞いと，内部の詳細な振る舞いを，設計意図として書き分けているという場合もある．内部の状態遷移に対してアスペクト関係ルールを定義するということは，内部の詳細な振る舞いまで全てモジュールの外側に公開するということを意味する．これが設計として妥当なことであるのかは，検討を要する．

本稿では，アスペクト指向モデルの動的意味を定義したが，これは意味を定義しているだけであり，実際にこのモデルを動作させる場合に，ルールが事前に解釈されてモデル上に展開されるのか(「プリコンパイル」されるのか)，アスペクト関係ルールの解釈系を実装し動的に解釈されるのか，までは規定していない．論理的な振る舞いとしてはどちらも変わらないが，コードに対応付けた場合はコード量や性能等，品質特性では差が出ると思われるので，目的に合わせて適切な方を選択すれば良い．

以上考察したように，本アスペクト指向モデリングメカニズムは，アスペクト関係ルールを用いることにより柔軟なソフトウェア構造の構成に有効であると考えられる．一方，アスペクト間の関係をルールとしてアスペクトから分離したことで，潜在的にはルールが非常に多くなる可能性もある．その場合，ルール同士に矛盾がないか，どのルールが同じ遷移に関係するものであるのかを，開発者が各アスペクトのモデルとルールを見ただけで理解することは困難を伴い，間違ったモデル化をする可能性も生じる．矛盾の可能性のあるルール(例えば複数の状態参照ルールがある同じ遷移の発火条件として絶対同時にはなることがない状態を指定している等)を提示し修正の要否を確認したり，同じ遷移を対象とする複数のイベント導入ルールを提示しその段階でそれらルールの順序付けができたりするようなツール支援も必要だろう．また，本モデリングメカニズムの特性を踏まえた上での望ましい設計のガイドラインとしてのデザインパターンなども必要と考えられる．これらの環境の整備も今後の課題である．

5. おわりに

本稿では，過去に我々が提案したアスペクト指向モデリングメカニズムについて，その動的意味を提案した．今後は，このメカニズムに基づき，アスペクト指向モデル駆動開発環境の完成を目指す．また，具体的なソフトウェア開発に適用し，有効性の評価をしたい．

参考文献

[1] https://www.eclipse.org/aspectj/
[2] Lewis, J. and Fowler, M.. Microservices. Mar. 2014. https://martinfowler.com/articles/microservices.html
[3] Noda, N. and Kishi, T.. Aspect-Oriented Modeling for Embedded Software Design. Proc. 14th Asia-Pacific Software Engineering Conference (APSEC 2007), Dec. 2007.
[4] Noda, N. and Kishi, T.. Aspect-oriented Modeling for Variability Management. Proc. 12th International Software Product Line Conference (SPLC 2008), Sep. 2008.
[5] 杉田郁人，野田夏子．アスペクト指向モデリング手法のためのモデル駆動開発環境の提案．情報処理学会研究報告，vol. 2017, no. SE-195, 2017.
[6] Tarr, P. et.al., N Degrees of Separation: Multi-Dimensional Separation of Concerns, Proc. 21st Int'l Conf. Software Engineering (ICSE'1999), 1999.

© 2018 Information Processing Society of Japan