Vol.2011-SE-173 No.10
2011/7/22

スケーラブルなモデリング技法に関する考察

岸　知二†

モデルベース開発、プロダクトライン開発、形式手法など、モデルを活用した開発形態では、従来以上に厳密、網羅的、詳細なモデルが求められ、結果としてモデリング作業が大規模、複雑化してきている。本稿では、開発で使われる複数のモデル間の整合性の問題に焦点を当て、部分的かつ弱い整合性を取りながら、徐々に整合性を高めていくアプローチについて、概念的な検討を行う。

On Scalable Modeling Techniques

Tomoji Kishi†

Recent software development techniques such as model-based development, product-line development and formal approaches require rigorous, exhaustive and detailed model, and it makes software models larger and more complicated. In this paper, we propose a modeling approach in which we firstly realize partial and weak consistencies among models, and then gradually enhance the consistencies along with the development lifecycle. We give a conceptual framework of the idea as our early research result.

1. はじめに

ソフトウェアの規模や複雑度が一層増す一方、開発期間の短期化、多様な利害関係者の開発への関与、製品系列開発の普及など、ソフトウェア開発は一層難しさを増している。そうした中、モデルベース開発 1)、プロダクトライン開発 2)、形式手法など新しい開発技術の研究や実務への適用が広がっている。こうした新しい開発技術に共通することは、より高度なマシン支援を想定していることであり、モデルベース開発におけるモデル変換､プロダクトライン開発における可変性の管理やプロダクト導出、あるいは形式手法における形式検証など、いずれもマシン支援なくしては実質機能しない。こうしたマシン支援を行うためには、コードだけでなく、要求、設計、構成など多様な情報を、形式性を持ったマシン処理可能な形で定義することが必須であり、広い意味でのモデリングが不可欠になっている。

UML3)などに代表されるモデリング技術はソフトウェアの開発手法において不可欠な技術であり、例えば構造化手法におけるデータフロー図の活用などは、1970 年前後に遡る。こうした開発手法は、状態遷移設計、データ中心設計、オブジェクト指向開発、アスペクト指向開発 4)などと多様化し、それらの中で状態遷移図やクラス図など様々なモデリング技法が使われてきた。従来こうしたモデルは、人間が読んで理解することを目的とすることが多く、より正確で標準的な文書の記法としての意味合いが強かった。したがって、モデルに対してプログラミング言語のような厳密性や網羅性は要求されないことが多かった。

しかしながら、上述したようにマシン処理を想定したモデリングが広がるにつれ、その性格は変貌しつつある。マシン処理をするためには、情報は厳密性や網羅性が要求され、また人間が読む場合以上の詳細度も要求される。そうでなければ、モデルをプログラミング言語に変換したり、可変性を解決して製品を導出したり、モデル検査技術によって精密に検証したりすることは困難である。その結果、モデルは従来以上に、大規模、複雑化する傾向にある。

本来モデリングは、規模の大きなソフトウェアの情報を抽象化し、本質的に重要な部分をできるだけ疑義なくコンパクトに表現するための技術であったはずであるが、現在ではそのモデリングそのものが大規模、複雑化し、コストのかかるものとなってしまってきている。こうした状況の中、より大規模、複雑なモデリングに対応できるモデリング技法への要求が一層高まっている。こうしたモデリング技法には様々な側面からの要求があるが、本稿では特に、モデル間の関係についての検討を行う。ここでモデル間の関係とは、例えば要求モデルと設計モデルといった、ひとつの開発の中で作られる異なった位置づけのモデル間の関係を意味する。モデルが厳密かつ詳細に

† 早稲田大学
Waseda University

なるにつれ、こうした異なったモデルを整合した形で維持することがより困難になっている。例えば要求モデルが変化したときに、それと整合する形で設計モデルや、実装モデルを作成する行為は、モデルが大規模、厳密、詳細になればなるほど困難かつ煩雑となり、現実には不整合なまま放置されたり、最悪の場合はモデルそのものが使われなくなったりする。本稿では大規模なモデリングを行う際のこうしたモデル間の関係の扱いに関して、概念的な整理や検討を行う。

2 章では、モデリングにおけるスケーラビリティとそれに関する課題、ならびに本稿での問題意識について述べる。3 章では、本稿におけるスケーラブルなモデリングに関する基本的なアプローチについて述べる。**エラー! 参照元が見つかりません。**章では、モデル間の関係について基本的な概念整理を行う。5 章ではモデル間の関係を俯瞰するための技法としてのモデルマップを提案する。6 章ではモデルマップを利用しながらスケーラブルなモデリングを行う際の戦略について検討する。7 章では関連する議論を行う。

2. モデリングにおけるスケーラビリティと課題

我々は大規模で複雑なモデリングを行うための技法の重要性を考え、スケーラブルなモデリング技法に関するワークショップを開催し、そこで様々な研究者や実務者と議論を行ってきた 5)6)。そこではモデリングにおける様々なスケーラビリティの問題が指摘された。以下はそうした問題の一部である。

- Size (modeling in the large)
- Complexity
- Team size (modeling in the many)
- Multiple concerns
- Multiple stakeholders
- Long life cycle
- Distributed environment

ここで指摘されているように、スケーラビリティは、単にモデルそのものが大規模で複雑になっているということだけでなく、そこで扱われている視点の多様化、開発における分散化や作業分担など多様な側面に関わった問題として捉えられる。またソフトウェアと同様、モデルそのもののライフサイクルが長くなっていることも重要な問題と指摘されている。モデリングの本質は抽象化であり、本来は規模の大きなソフトウェア開発を行うために、コードよりも抽象度高く開発上の情報を表現したり理解したりするために導入されたものと捉えられるが、現在はそのモデリングそのものがスケールの問題に直面している。さらに前述したようにモデルをマシン処理する要求が高まっており、そこではより厳密、網羅的、詳細なモデルが必要とされ、この傾向は一層強まっているといえる。

本稿では、こうした様々なスケーラビリティに関わる課題に対応できるモデリング技法をスケーラブルなモデリング技法と呼ぶ。スケーラブルなモデリング技法の実現のためには、様々な問題を改善する必要がある。ワークショップでは、例えばモデル化される対象の理解、モデル化対象に関わる情報のマイニング、モデルの構築や再構成、モデルの視覚化、モデルの段階的な変換、モデル進化など、様々な側面について指摘や議論がなされた。

これらの問題はいずれも重要であるが、本稿では特にソフトウェア開発に必要とされる様々な位置づけのモデル間の一貫性や整合性をどのように維持管理するかという課題に焦点を当てて考察を行う。要求モデルと設計モデル、アーキテクチャモデルと詳細設計モデルなど、開発の中では様々な異なったモデルが利用され、それらの間には多様な関係が存在する。例えば設計モデル中の特定の設計構造は、要求モデル中の特定の要求項目の実現のために存在するのかもしれないし、詳細設計モデル中のある構造はアーキテクチャモデル中の構造を特殊化したものかもしれない。ソフトウェアの進化に沿ってモデルも進化し、モデルの進化に沿ってこうしたモデル間の関係も整合性のある形に維持管理されなければならない。例えば要求モデルに変更があれば、設計モデルも変更され、その対応関係も変更されうる。しかしながらこうした維持管理はモデルが大規模化し複雑化するほど困難になる。

3. モデリングアプローチ

モデリングにおいて、モデル間の関係を一貫性のある状態に保つことは重要であり、それが妥当なコストでできるならば問題はない。しかしながら現実にはそれは困難である。そもそも開発そのものが常に要求や環境の変化にさらされており、開発の過程において様々な変化が常に発生するため、それに即座に対応してモデル間の関係を整合性のある形にすることは現実的ではない。また未知のリスクに備えられるように例えばインクリメンタルな開発を採用するなど、開発スタイルそのものも変化を前提としつつある。ウォーターフォール型の開発では、原則的に一度定義された上流の情報は変化させず、徐々に下位の情報を生成するという立場であったが、そうした開発スタイルは今日では機能しない。

こうした状況を考えると、大規模なモデル全体を常に一貫性のある形に保つということ自体が非現実的なことと考えられる。構成管理などの世界においても、成果物間の対応関係などのリンク情報を維持管理することは、現実のソフトウェア開発の中では困難である。しかしながら無戦略に一貫性の維持を放置しそれをアドホックに受け

入れると、モデルのマシン処理を行うことが無意味になる。整合のとれていない要求モデルと設計モデルを突き合わせて形式検証を行っても、その結果は何も有益な情報をもたらさない。したがって、全体の整合性保持が困難であるにせよ、整合性をどのように維持するかということに関して､方針や考え方を明確化することが必要である。

ここまでで議論したように、本稿では以下を前提として考えることとした。

前提：スケーラブルなモデリングにおいては、モデル全体を常時整合性のとれた状態にすることは困難である。

それを前提とした上で、次善の策として以下を目指す。

次善の策：部分的かつ弱い整合性でもそれなりに運用できるモデリング体系をつくる。そのうえで、必要に応じて整合性の範囲や整合性の程度を漸次改善することを可能とする。

直感的には、整合性をとる範囲を限定し、また場合によっては弱い整合性を取ることを許容することで、その時点時点において最低限必要な整合のみを取ることを許容するというアプローチである[a]。もちろん整合性の範囲や整合性の程度が限定されることで、モデル全体が完全な整合性を持つ場合に比べて、モデルをマシン処理して得られる情報や結果は一般に弱いものとなるが、そもそも完全な整合が困難という前提であるから、次善の策としてそれを認める。ただしこの場合、その範囲や程度をどう把握するか、という点が問題となる。そうした範囲や程度を把握するためには、様々なモデルがどのような関係を持ち、それらがどういう依存関係になっているかをマクロに理解し、どういう範囲をどの程度まで整合させるかという戦略付けを行うことが重要と考える。次章以下、こうしたアプローチについて、より詳細に検討する。

4. 基本概念

アプローチの説明をするために、モデルやモデル間の関係について整理する。なおここで議論するモデルやモデル間の関係は、前章で述べたアプローチを行う上での戦略付けの議論に使われるものであり、一定の抽象度を持った疎粒度のものである。

4.1 モデル

本稿では開発で使われる情報のうち、なんらかの形式性（シンタクスとセマンティクス）に基づいて表現された情報をモデルと呼ぶ。典型的には UML などのモデリング記法に基づいて表現された情報を想定するが、その開発においてその情報が形式性を持っていると考えられるなら、具体的にどういうものをそれに含めるかは当事者の判断で決めればよいと考える。プログラミング言語や OCL による制約記述などをモデルと捉えて構わないし、マシン処理可能でなくても一定の形式性を持つものをモデルとして捉えても構わない。

こうしたモデルには様々なものがあるが、大きく以下の 3 つの観点からそれを特性づけることができると考える。

- 開発の中での位置づけ：そのモデルが開発においてどういう意味合いで作られたものであるか。例えば要求モデルであるか設計モデルであるか、アーキテクチャモデルであるか詳細な設計モデルであるかなど。どういう意味合いのモデルが必要かは開発によって異なる。
- モデルのタイプ：モデルによってその表現されていることがらが例示なのか一般的な記述なのか等、モデルの表現している内容の種別。典型的には、以下の分類を用いると、モデル間の関係の検討に有用と考える。
  - 例示モデル：対象の構造やふるまいなどの、特定のインスタンスを示すモデル。例えばオブジェクト図やシーケンス図などであらわされるモデルなど。
  - 条件モデル：対象の構造やふるまいなどを直接的に記述するのではなく、その構造やふるまいに対する制約や条件を示すモデル。OCL による不変表明の記述など。
  - 一般モデル：対象に構造やふるまいなどを、特定の視点から一般的、網羅的に示すモデル。クラス図やステート図などによる記述など。
- 具体的な形式：実際に用いられる形式性を示す。クラス図などの図法、用いるプロファイルの種類等。

4.2 モデル間の関係

上述したモデル間にはいろいろな関係がある。関係を考えるために、開発の中でモデルがどのように使われうるかを、定義と検証という典型的な行為を例に挙げて以下に示す。

- 定義：要求モデルに基づいて設計モデルを定義するという状況に照らして、これら二つのモデルの関係を考えてみる。例えば例示モデルとして与えられた要求モデル（利用方法の典型的な状況をシーケンス図として定義する等）に基づ

a 分散データ処理における可用性優先(basically available)や結果一貫性(eventual consistency)など 7)を参考にした考え方である。

いて、一般モデルとしての設計モデル（クラス図とステート図によって表現する等）を定義する状況を考えてみる。この場合は、要求モデルで示されていることがらを設計モデルが満たさなければならない。さらに要求モデルにおいては例示されていない状況にも設計モデルは対応できなければならない。一方、別の状況として、一般モデルでの要求モデルに基づいて、一般モデルとしての設計モデルを構築することを考える。この場合は仕様上のふるまいは網羅されているので、設計モデルの表すふるまいは少なくとも仕様上の抽象度で眺める限りにおいては仕様上のふるまいどおりでなければならない。

- 検証：ひとつの設計モデルに基づいて（それを検証性質として）、他の設計モデルを検証する状況に照らして、これら二つのモデルの関係を考えてみる。例えば条件モデルとして与えられた設計モデル（設計において成立しなければならない不変表明を表す等）に照らして、一般モデルとしての設計モデル（ステート図で表現される等）がその不変表明を満たしているかどうかを検証する状況を考える。この場合は、後者の設計モデルはあらゆる状況で前者のモデルを満たさなければならない。一方前者が例示モデルで与えられるとすると、後者は例示された局面では前者のとおりでなければならないが、例示されていない状況も包含しなければならないので、条件モデルの場合に比べて弱い検証性質しか与えていないことになる。

上の例での違いを議論するために、モデル間の関係を以下のように整理する。

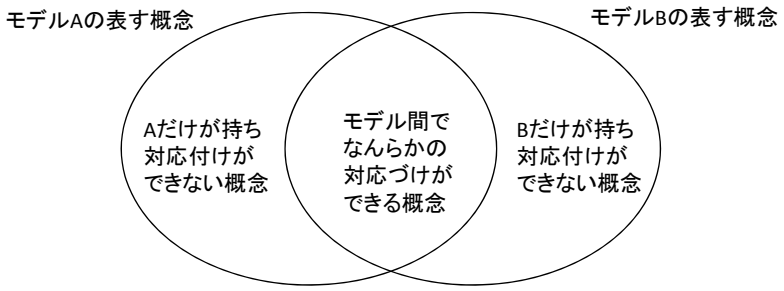

図 1　二つのモデルと、モデルの表す概念の関係

図 1 は、二つのモデル A,B が持つ概念の関係を模式的に表したものである。モデル中の概念の中には、他のモデルの概念と何らかの対応付けができるものとできないものとがある。例えば要求モデルにおいて「システムが警告を出す」という概念があったときに、それは設計モデルにおける「エラーメッセージを出力する」という概念と対応付けることができる。一方、例えば要求モデル中に記述される人間系の背後情報や、設計モデル中に記述される実装技術依存の情報などは対応付けができない[b]。本稿でモデル間の関係を定義する際には、なんらかの対応づけができる概念の部分のみに焦点を当て、以下のように分類する[c]。なお対応付けができる概念がなければ関係は定義されない。

- 全域関係：一方の概念のインスタンスがすべて他方の概念のインスタンスと対応付けられる状態をいう[d]。上述の例では、一般モデルとしての要求モデルと一般モデルとしての設計モデルは全域関係にあるという。
- 部分関係：一方の概念のインスタンス一部に対してのみ対応する概念のインスタンスがあり、一部に対しては対応概念のインスタンスがない状態をいう。上述の例では、例示モデルとしての要求モデルと一般モデルとしての設計モデルは部分関係にあるという。
- 包含関係：部分関係のうち、一方の概念のインスタンスは必ず他方に対応付けられるが、逆は成立しないとき、必ず対応づけられる側のモデルが他方に包含されているという[e]。

こうしたモデル間の関係を検討する際には、両モデルが、前述した例示モデル、条件モデル、一般モデルのいずれに属するかを考えると理解がしやすくなることが多い。両者が一般モデルの際は基本的に全域関係であり、一方が一般モデルで他方が例示モデルの場合は、一般モデルが例示モデルを包含する包含関係となる。但し、他の場合は必ずしも関係は機械的に導出されない。例えば条件モデルの場合、その条件が特定の状況に関わる条件のみを与え、他の状況に関しては無関心(don't care)であるのか、他の状況ではその条件が成立してはならないといっているのかによって、関係は異なるので、意味に立ち入った検討が必要である。

さらに、対応付けにおいて、そのモデルだけが持っている概念を考慮したときに、対応付けられた概念のインスタンスが複数のインスタンスに対応するかどうかを検討する。例えば要求モデル中の「満空状態」という概念が設計モデル中の「受付バッフ

b 本稿では対応付けとは、ソフトウェア開発において関わる情報や定義が存在するという程度の緩い意味で使っている。概念の一層の正確化は今後の課題である。

c 対応付けが不可能な部分（上述した非ソフトウェア部分や実装技術依存部分）は、他方のモデルにとっては関心事でないからである。モデルは関心事が異なれば記述範囲や記述方法が異なるため、本来関心事が異なりモデルで記述されていなかった部分に関する情報の過不足は議論しない。

d 例えば要求モデル中の「動作モード」と、設計モデル中の「モードを保持する変数値」という概念が対応付けられるとする。動作モードはインスタンスとして様々なモードを持ち、変数値はインスタンスとして様々な値を持つ。両インスタンスがもれなく対応づく場合もあるが、モードと対応づけられないような変数値があるなど、もれなく対応付けられない場合もある。

e 例えば、例示モデルと例示モデルの間などでは、部分関係だが包含関係でない状況が起こりうる。

ァ状態」という概念と対応し、前者の「受付可能」というインスタンスが、後者の「バッファが満でない」というインスタンスと対応付けられたとする。しかし内部の実装上、バッファ中のデータ数がある数以上の場合と未満の場合ではその処理が異なっていた場合、「バッファが満でない」というひとつのインスタンスが、設計モデル中では複数のインスタンス（「バッファ中のデータがある数以上」「バッファ中のデータがある数未満」）に対応することになる(図 2)。

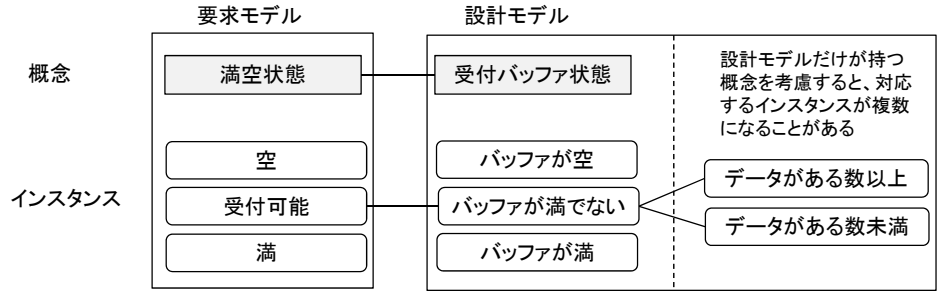

図 2　インスタンスの多重度が増える例

要求モデルに照らして設計モデルを検証するような状況では、この対応付けが重要となり、「バッファが満でない」ことに対応する複数のインスタンスのすべてを検証するのか、そのうちの一部を検証させるのかによって、検証のコストと質が変わりうる。直感的には、二つのモデルの一方がより抽象的であれば、抽象側のひとつのインスタンスに、具象の側の複数のインスタンスが対応することが多くなる。

5. モデルマップ

ソフトウェア開発で用いられる様々なモデル間には、前章で定義した関係のいずれかが成立する。それを俯瞰するための図法としてモデルマップを提案する。

モデルマップは、モデル間の関係を疎粒度のレベルで概観するためのグラフ表現の図であり、以下のモデル要素によって構成される[f]。

- モデル：モデルは四角であらわす。モデルは 4.1 で述べたようにあくまで開発者が議論したい範囲、粒度で捉える。必要に応じて、位置づけ、タイプ、形式などを表明として記述する。
- ソース：モデルとしての形式性は存在しないが、ソフトウェア開発に必要な情報が存在すれば、それを破線の楕円で表し、それが何を表すかを表明として記述する。
- モデル間関係：モデル間には以下の関係を定義することができる。
  - 全域関係：実線で示す
  - 部分関係：矢印で示す。包含関係の場合には、被包含側に矢印を付ける。包含関係でない部分関係の場合は両端に矢印を付ける。

  モデル間関係では、さらにクラス図の多重度の表記を用いて、一方のモデルだけが持っている概念を考慮したときに概念インスタンスが複数になるかどうかを示す。
- 依存関係：モデルとソースの間に、破線矢印で示す。

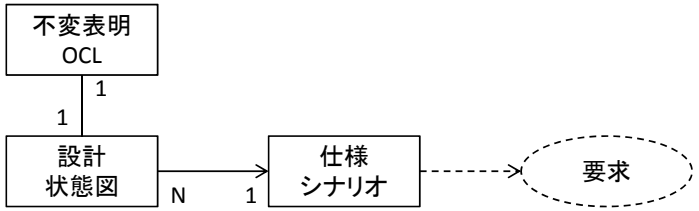

図 3　モデルマップの記述例

図 3 はモデルマップの記述例である。シナリオで書かれた仕様（例示モデル）と、状態図で書かれた設計（一般モデル）の間には包含関係があり、設計に対する不変表明を示した OCL(条件モデル)は設計において常に成り立つ必要があるので全域関係がある。仕様モデルは設計モデルより抽象的であり設計モデル側の多重度が N となっている。不変表明は設計と同じ抽象度で書かれており、多重度は 1 となっている。なお仕様は非形式的に記述された要求に基づいて記述されたものであることがソースと依存関係で示されている。

ソースは一般的に断片的かつ不完全なものであることが多いので、ソースに依存して定義されたモデルはそうした側面を補完したり修正したり解釈したりして作られることが多い。したがってモデルに含まれる情報の根拠や正しさを検討する際には、こうしたソースとモデルの接点で何が行われていたかを理解することが有用なことが多い。ソース自体はモデルではないが、モデルの活用を考える際には重要な側面であるため、モデルマップでは記述する枠組みを用意した。

f 簡単な記法なので、本稿では直感的な説明で示す。表明部分の記述方法などは今後明確化する必要がある。

6. モデリングの戦略

3 章で、スケーラブルなモデリングのために、部分的かつ弱い整合性でもそれなりに運用できるモデリング体系をつくるアプローチをとることを示した。以下、モデルマップを利用しながら本アプローチについて検討する。

6.1 整合性の範囲と強さ

整合性の範囲については、モデルマップ中に示されるすべてのモデル間関係の両端のモデルが整合したとき、最も広い範囲が整合化したと考える。一方、整合させるモデル間関係が少なくなるほど、その範囲は部分的になると考える。

整合性の強さに関しては、全域関係の整合化の方が、包含関係の整合化よりもより広範な状況について整合化を行っていると考え、全域関係の整合化は部分関係の整合化よりも強く、部分関係の整合化は全域関係の包含化よりも弱いと考える。包含関係でない部分関係と、包含関係の強さの比較議論はしないが、整合化作業は一般に包含関係の方がやりやすいと考えられるため、本モデリングにおいては包含関係がより望ましいと考える。さらに多重度が N の場合、その整合性をどこまでとるかにより、強さが変わると考える。例えば複数のインスタンスが対応する際に、そのすべてを整合させるのか一部を整合させるのかによって整合性の強さが変わると考える。

なお整合化において、整合させるべき相手のモデルが、より大元のモデルと整合している場合、その相手のモデルの信頼性が高いということにする。信頼性は連鎖しており､大元のモデルの信頼性が低ければ整合させても信頼性は十分ではないと考える。ソースへ依存している情報は信頼性について十分に検討すべきである。

6.2 アプローチ

前項を踏まえて本稿で提案するアプローチを再度説明すると、モデルマップ中の全モデル間関係を対象にせず、その中でいくつかのモデル間関係にのみ注目し、そのモデル間関係の整合化のみを行うという方法である。またその時に、全域関係の整合化を、部分関係、包含関係の整合化に弱める、多重度 N の部分の整合化を弱めにするなどすることで、その整合化をより低コストに行うことを考える。

ただしそうすることでペナルティが発生する。例えば部分的にすることにより、本来はあるモデルと整合化させる前に、そのモデル自体を、大元となっているモデルと整合させることが望ましいのだが、その作業を省略することにより、整合化させるモデルの信頼性が低くなることが起こる。あるいは弱い整合性を取ることで、整合の質が低下する。本アプローチでは、そうしたペナルティの発生自体は仕方がないと考えるが、どういうペナルティが発生しているかをできるだけ明示的に認識することが重要と考える。現時点での整合性の弱点が理解できていれば、その後の開発や検証の過程で、それを強化することが可能となるからである。

6.3 モデリングアーキテクチャ

ある開発において作られるモデルの全体像をモデルマップで表すと、本稿の提案するアプローチに適したモデル間関係なのかどうかをある程度判断することができる。例えば包含関係でない部分関係が多出するようなモデル構成（例えば例示モデルばかりが作られ、例示の間の関係が不明確な状況など）は、整合化の戦略がたてづらい。あるいは、重要なモデルがソースに直接依存している（例えば様々な暗黙の仮定、前提、経験に基づいて設計がなされているなど）は、整合化作業がやりづらい（詳細不明のフレームワークの利用を想定した設計モデルを形式検証しようとしてもできないなど)。

本稿では、モデルマップを、開発において利用されるモデルとそのモデル間関係で表されるモデリングの疎粒度の構造表現と考え、そこで表される構造をモデリングアーキテクチャと呼ぶ。この用語を使うなら、部分的かつ弱い整合化に適したモデリングアーキテクチャとそうでないモデリングアーキテクチャがあり、ソフトウェア開発においては、そのスケールに適したモデリングアーキテクチャを構築することが重要であると考える。より具体的に言えば、以下のような特性を備えたモデリングアーキテクチャがよりスケーラブルであると考える。

- モデルマップが明確に書ける：モデルマップを書くためには、どのようなモデルを利用しているのか、そのモデルがどのような位置づけ、タイプ、形式なのかを理解し、かつそれが他のモデルとの間にどういう関係を持ち、その関係が全域関係か部分関係かさらには包含関係か等を明示的に理解する必要がある。こういうことを明示的に考えずに漠然とモデリングを行っていてはそもそも整合性議論そのものが無意味となる。例えば形式検証をする際に、与えられた検証性質と検証対象のモデルとが全域関係なのか部分関係なのかを理解せずに検証を行っても、その結果を正しく理解することはできない。
- 部分的な整合性が意味を持つと考えられるサブグラフが存在する。例えばその開発において、比較的安定的であったり、信頼性が高いとみなすことができたりするモデルやソースがあり、そこを出発点とした閉じたサブグラフがあるなどすればその部分だけの整合性を考えることが、実質的に有意義となる。逆の言い方をすれば、そういうサブグラフをどこに作ることが現実的であるかを考えながらモデリングアーキテクチャを設計することが重要と考える。
- 強い整合性をより弱い整合性で置き換えるパスが存在する。例えば図 4 は仕様モデルと設計モデルの一例である。仕様がシナリオで与えられ、それを一般化した状態図で表し、さらに設計の状態図を構築したとする。仕様シナリオが変更された際、本来はそれに基づいて仕様状態図の整合をとり、それに照らして設計状態図の整合をとる必要がある。これを仕様シナリオに照らして設計状態図の整合をとるというパスも検討可能である。もちろんそれによってとられる

整合性は弱くなるが、状況によっては有用な戦略である。本アプローチはこうしたパスを検討し活用することを意図している。なお、多重度が N の部分は、その部分で整合性をとる概念数を減らすことで整合化コストを減らすことが期待される。

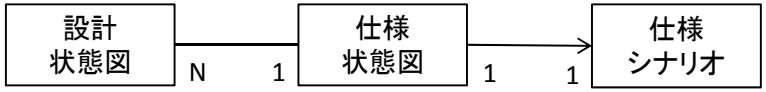

図 4　設計モデルと仕様モデルのモデルマップ例

このように、モデルマップを活用することで、よりスケーラブルなモデリングアーキテクチャの検討を行うことができると考える。整合性の範囲や強さは何段階かのレベルで捉えることができるため、その時点での整合化に割けるコストと、開発上最低限必要な整合のレベルやモデルの信頼性に応じて妥当な整合化を行い、その後必要に応じて漸次整合のレベルを高め信頼性を向上させる、段階的整合化を行うことを意図している。

6.4 モデリングアーキテクチャとソフトウェアアーキテクチャ

モデリングアーキテクチャはソフトウェアのアーキテクチャ 8)とも強く関係している。アーキテクチャは開発対象の製品や製品群にとって重要な品質特性を達成するための基本的な構造、あるいはそのための設計判断の集合と考えることができる。図 5 はアーキテクチャと製品の要求モデル（シナリオ）と設計モデル（状態図）の関係をモデルマップで示したものである。一般的にはここに示されるモデル間関係の中で一番重要なものは､アーキテクチャ設計とアーキテクチャへの要求の間の関係であろう。この部分の整合の度合いが高ければ､後の関係の整合化は相対的に容易だからである。このように、ソフトウェアのアーキテクチャを考えることで、どの部分の整合化を行うことがより重要であるかという判断に利用できるし、逆に言えばソフトウェアをどのようなアーキテクチャで捉えるかによってモデリングの戦略が変わってくると考えられる。

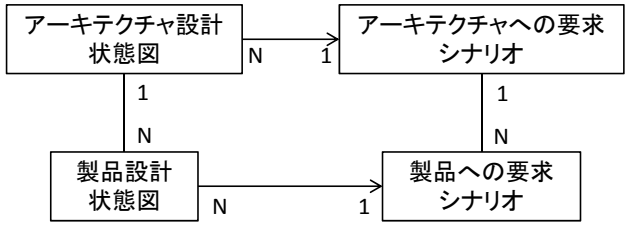

図 5　アーキテクチャのモデルと製品のモデルのモデルマップ例

7. 議論

本稿では、スケーラブルなモデリング技法に関して、特にモデル間の関係に注目して、マクロなモデリングのアプローチについて、概念的、基本的な検討を行った。本提案は、モデリングの個別のテクニックではなく、より疎粒度なモデルの利用方法や位置づけの検討あるいは設計に関わるものである。

モデリングが大規模、複雑化するにつれ、現実問題としてはモデル間の意味的な整合性をとることがきわめて困難になってくる。したがって、あらゆる時点でモデル全体が整合性のとられた状態にあることを期待することが難しくなってきている。しかしながら、その整合化の作業をアドホックに進めていては、モデル化による本質的なメリットが享受できない。そこで本稿では、モデルマップという簡単な図でモデリングアーキテクチャを捉え、部分的かつ弱い整合化を行う戦略付けの支援をすることを提案した。モデルマップからは本稿で指摘した以外にも様々な戦略付けの情報を読み取ることができるが、本稿では典型的でわかりやすい情報の提示にとどめた。

スケーラブルなモデリングを行うためには、個別のテクニック以上に、個々のモデルの位置づけや信頼性の理解が不可欠になる。例えばモデル検査技術による設計検証を考えてみると、ソフトウェア開発者は、ともすればモデル検査エンジンの詳細な利用方法や検証技法にとらわれ、検証対象のモデル化やその検証性質の位置づけや信頼性について、十分な配慮をしない場合がある。設計対象が小さければ直感的にモデルを作っても、ある程度有用な検証ができるかもしれないが、規模が大きくなるとそれは困難になる 9)。せっかく高度なテクニックを利用し、手間をかけて複雑なモデルを検証したのに、いったいそこからどういう結果が得られたのかが不明確という状況になりかねない。モデルマップはこうした大規模なモデリングにおける戦略付けを支援するものと考えている。

なお本フレームワーク検討にはプロブレムフレーム 10)などの考え方や図法も参考

にした。

8. おわりに

本稿では、スケーラブルなモデリング技法に関す概念的かつ初歩的な検討を行った。本稿で提案する段階的整合化は、モデル駆動における段階的なモデル変換、構成管理における段階的な構成導出など､複数の局面で活用できる考え方であると考えている。今後これを実現するための具体的な技術や手法を整理し、実際の有効性について踏み込んだ検討をしていきたい。

参考文献

1) www.omg.org
2) www.splc.net
3) www.uml.org
4) www.aosd.net
5) Natsuko Noda and Tomoji Kishi, New Challenge of Scalable Modeling, 2nd Workshop on Scalable Modeling Techniques for Software Product Lines (SCALE 2010), the second proc. of SPLC2010, pp191-192, 2010.
6) Tomoji Kishi and Kyo-Chul Kang: Scalable Modeling Techniques for Software Product Lines (SCALE 2009), proceedings of SPLC2009, p299, 2009.
7) Eric A. Brewer, Toward Robust Distributed Systems, keynote, PODC, 2000.
8) 岸知二, 野田夏子, 深澤義彰: ソフトウェアアーキテクチャ, 共立出版, 2005.
9) 岸知二: モデル検査のための設計モデル構築手法に関する考察, 情報処理学会 SIGSE, vol.2010-SE-168, No.9, pp1-6, 2010.
10) Michael Jackson, Problem Frames, ACM Press, 2000. 邦訳 : 榊原監訳, プログラムフレーム, 翔泳社.