社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－SE－159 (24)
2008／3／18

形式手法Alloyを用いた図式モデルのための段階的検証方法

寺中 慎介[†], 村上 祥平[††], 小飼 敬[†††], 上田 賀一[††]

[†]茨城大学大学院 理工学研究科　[††]茨城大学 工学部　[†††]茨城工業高等専門学校

近年，ソフトウェアに対する信頼性や安全性に対する関心が高まり，形式手法が注目を集めている．形式手法に対する敷居の高さを解決するために，これまでにZ言語で記述された図式モデルを対象に形式手法に則った検証が行える形式手法支援ツールを実現してきた．本研究では，この図式モデルを対象としたより実用的な形式仕様記述支援ツールについて模索する．方針として，既存の処理系としてライトウェイトな形式手法が実現できるAlloyに着目し，図式モデルに対する段階的な検証方法を提案する．また，検証例としてDFDの階層化を行い，制約記述の方法や検証の性能評価について考察した．

A Phased Verification Method for Diagrammatic Model Using Alloy

Shinsuke TERANAKA[†], Shohei MURAKAMI[††], Kei KOGAI[†††] and Yoshikazu UEDA[††]

[†]Graduate School of Science and Engineering, Ibaraki University
[††]Faculty of Engineering, Ibaraki University
[†††]Ibaraki National College of Technology

In late years, the concern for reliability and safety to the software rises, and the formal method attracts attention. To solve the height of the threshold to the formal method, we had realized the formal method supporting tool, which could perform the verification of the formal specification for a diagrammatic model written in Z language. In this study, it seeks for a more practicable formal specification supporting tool for this diagrammatic model. As the policy, we pay our attention to Alloy which can realize a light-weighted formal method as an existing processing system, and propose a phased verification method for a diagrammatic model. Moreover, we hierarchize DFD as the verification example, and we consider the method and the performance of the verification of the restriction specification.

1 はじめに

近年，ソフトウェアの信頼性や安全性に対する関心が高まり，形式手法が注目を集めている．形式手法とはソフトウェアの仕様を数学的に表現した上で検証を行い，モデリング時に矛盾を見つける方法である．また，ソフトウェア工学の道具としての形式手法[1]では，数理論理学に基づく言語ならびに証明技法などからなる基礎理論を持つことに加えて，設計手法，ドメイン知識などが関係する総合的な技術として捉えられている．形式仕様記述言語としては，Z言語[2]，object-Z[3]といったものが挙げられる．

本研究室ではこれまでに形式仕様記述に関する研究[4]が行われてきた．その研究では，ソフトウェア開発で作成される仕様書で使用される様々な図式モデルに対し，これらのモデルが不完全でなく曖昧さがないようにするために形式的な仕様記述の作成を支援するツールを提案している．実現方法としては，メタ階層アーキテクチャを導入することで，メタメタモデルを使ってメタモデルを作成，メタモデルを使ってベースモデルを作成するという手法を用い，様々な種類の図式モデルの仕様を形式化することを可能としている．これらの作成した形式仕様記述はノードとアークに基づいた集合を扱えるようZ言語で記述され，検証を

するためにZ言語の処理系であるZANS[5]と連携することで図式モデルの検証を実現している．しかしながら問題点として，図式モデル固有の操作の形式仕様記述の作成が困難であることとZANSの能力に限界があると指摘している．

これらの背景から，本研究ではより実用的な形式仕様記述支援ツールについて模索する．方針として，形式手法Alloy[6]に着目し，図式モデルのための段階的検証方法を提案する．Alloyは形式手法におけるモデル検査器ともいわれ，充足可能性問題(SAT)でインスタンスや反例を探索し，利用者が欲するモデルを提示する．これらの機能によってモデルの自動検証も可能となる．

本論文の構成は以下のようになっている．2章では，提案手法である段階的検証法の概念と実現方法について述べ，3章では，DFDの階層化を例に提案手法について考察する．4章でまとめを示す．

2 段階的検証方法

2.1 概要

本研究の概要は図1のようになる．本研究では形式手法Alloyを用いて図式モデルを記述し，実行する．具体的にはAlloy Editorでメタメタモデル，メタモデル，ベースモデルをAlloy記述で作成していくことを想定している．Alloyでは記述したモデルをSAT問題への入力として捉え，SAT solverで解を探索する．

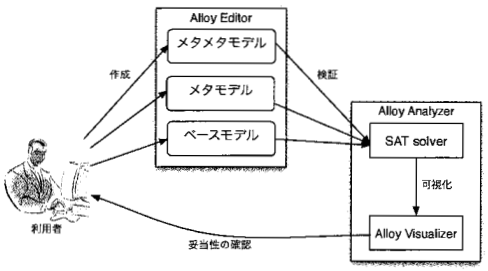

図1: 本研究の概要

また段階的検証法のイメージは図2のようになる．これは各レベルのモデルをAlloyのrunコマンドやcheckコマンドを切り分けることで実現する．利用者はrunコマンドを実行することで，下のレベルのモデルのインスタンスを確認し，そのインスタンスが妥当なものであるかを見る．そして妥当でない場合は上のレベルの制約記述が足らないことがわかり，修正する．つまり，runコマンドは妥当性の確認(Validation)として機能する．checkコマンドは基本的に制約記述が正しいかどうかを保証するために実行する．これは上のレベルの制約に反していないかを見ることで，下のレベルのモデル記述が正しいことを保証する．つまり，反例がなければ，その記述が正しいことが保証される．checkコマンドは検証(Verification)として機能する．また，上のレベルと下のレベルの切り分けは，メタメタレベルとメタレベル，メタレベルとベースレベル，として切り分ける．

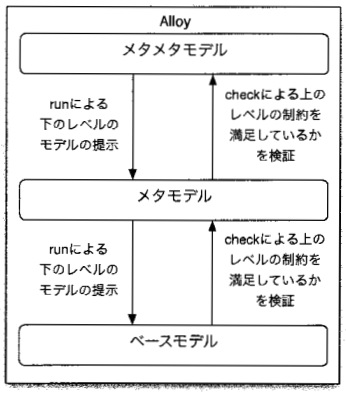

図2: 段階的検証法のイメージ

2.2 メタ階層に基づいた図式モデル

メタ階層に基づいた図式モデルでは，いくつかのレベルにモデルを分けてモデルの意味を定義している．これは図3のようにメタメタモデル，メタモデル，ベースモデルの3つのレベルのモデルが存在し，1つ上のレベルのモデルが下のレベルのモデルを記述し，お互いはメタの概念の関係で結ばれている．そして3つのモデルに対応した形式仕様記述があり，その3つの形式仕様記述全体で1つの図式モデルの意味を表す．

2.3 各レベルのモデル表記と記述方法

メタメタモデルが持つ意味を図4に示す．このモデル分割では，定義されたモデルが簡潔でわかりやすいものを目指すために，ノード・アークモ

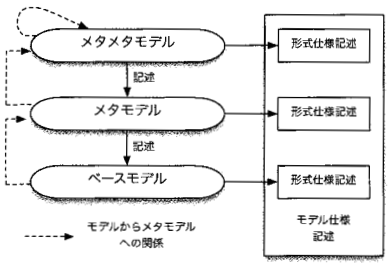

図 3: メタ階層アーキテクチャによるモデル仕様の分割

デルというシンプルなモデルを基本とする．表記法は図 5 のようになる．これによって，モデルの構成要素とその要素間のつながりが表現可能となる．メタメタモデルではノード・アークモデルを基に ER 図として定義されている．メタメタレベルの要素では，ENTITY，RELATIONSHIP，FIELD，CONTAIN という 4 つの要素があり，ENTITY はノードの要素として，RELATIONSHIP はアークの要素として定義される．また大規模な図式モデルにも対応できるように，モデルを構造化するために必要な FIELD の概念を追加する．FIELD は ENTITY，RELATIONSHIP，FIELD を CONTAIN することができる． DFD（データフロー図）を

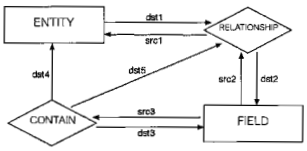

図 4: ER モデルベースのメタメタモデル

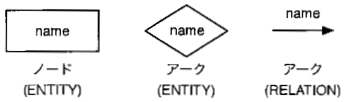

図 5: ノードとアークの表記

例に取り上げて，実際のメタモデルの定義を示すと図 6 のようになる．このメタモデルでは，DFD を作成するのに必要となる構成要素，PROCESS，DATASTORE，ACTOR，DATAFLOW をメタメタモデルの ENTITY から作成し，これらの要素の関連をメタメタモデルの RELATIONSHIP を使って関連付けしている．

図 6 のメタモデルを使って，実際の DFD がベー

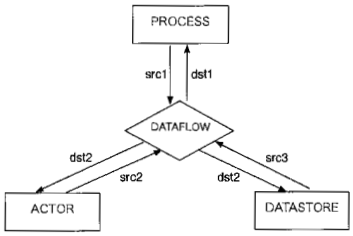

図 6: DFD のメタモデル

スモデルとして作成できる．ベースモデルの例を図 7 に示す．DFD の例からもわかるように，メタモデ

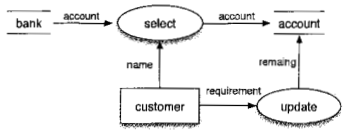

図 7: DFD のベースモデル

ルの定義において，ENTITY から作成された構成要素にはノードとしての役割を持つ要素が存在する．つまり，PROCESS，DATASTORE，ACTOR は DFD ではノードタイプの要素，DATAFLOW はアークタイプの要素となり，ベースモデルではそのように働いている．また，これはメタメタモデルにおいても同様で，ENTITY，FIELD はノードタイプ，RELATIONSHIP と CONTAIN はアークタイプの構成要素として定義されている．したがって，DFD のメタモデルの中で，矩形で表記されている要素は ENTITY から作成され，矢印で表記されている要素は RELATIONSHIP から作成されている．このようなメタな制約のイメージを図 8 に図示する．図式モデルによるメタ制約に従った検証はこのようなメタな関係を考慮した時に，メタメタレベルでの要素間の関係またはメタレベルでの要素間の関係が記述可能であるかを見ることでメタレベルやベースレベルで整合性が取れているかを検証する．

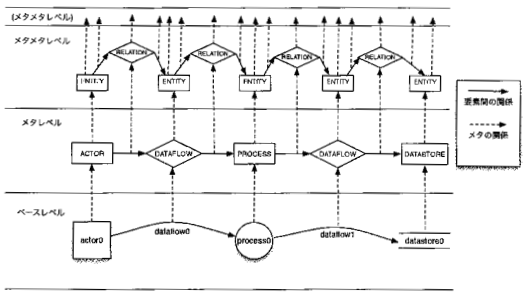

図 8: メタ制約のイメージ

2.4 図式モデルの検証方法

図式モデルの作成方法として以下２つの方法が考えられる

- ゼロからモデルを作成する
  ZANS のような，Z 言語の処理系はこの作成方法を用いて検証される．Alloy でも操作スキーマを実行する前に検証していくという方法は同等である．これは操作スキーマでは Alloy 記述で記述された述語 pred 内にある記述を run コマンドで実行されるため，モデル作成時の制約を充足するモデルが常に作成されることになる．つまり，run コマンドによって探索される解は図式モデルのインスタンスであり，それは正しさを保証された図式モデルであるともいえるため，作成時に関して常時検証されていると言える．

- 外部ツールからのモデルを変更・作成する
  Alloy 以外の外部のモデリングツールで作成された図式モデルを Alloy 上で検証することを想定した場合，Alloy で検証できる項目は，図式モデルの制約，つまりメタモデルおよびメタメタモデルの制約に則って作成されているかどうかを検証する．この検証時に，図式モデルの要素に原因箇所を示すフィールド (例えば cause) や不整合となる原因の種類 (例えば kind) といったものを加えることで，不整合時に図式モデルのある要素で問題箇所を特定し，かつその原因の種類に応じて修正することが可能となる．なお，このような問題箇所の指摘はノードではなくアークのみに持たせることが一般的ではないかと考える．ノードのみの指摘では，図式モデルのセマンティクスは，その図式モデルを利用しているプロジェクトのドメインのセマンティクスについて言及してしまうことになるため，これは本研究のメタ階層アーキテクチャを用いた段階的検証方法では検出できないと考えられる．すなわち，アークがあって初めてノードの問題指摘が可能となるのであって，これはアークのコネクション制約に則っているかどうかを検証すれば良いという意味で，アークのみの問題指摘にとどまることが有効ではないかと考える．

上記の検証方法を用いるためには，Alloy 記述でメタモデルおよびベースモデルをモデル作成者が記述することになる．そしてメタモデルまたはベースモデルのノードやアークはシグネチャ(alloy 記述では sig) として記述され，そのときの制約は不変条件 (alloy 記述では fact) として記述される．

実現方法として，状態スキーマ ID に validity というフィールドを持たせることで整合であるか不整合であるかの結果を持つことができる．また，アークタイプの制約を述語 (Alloy でいう pred) で記述することによって，run コマンドで問題箇所の指摘および修正方法と修正された図式モデルの提示が可能となる．

2.5 検証するための記述方法

ノードやアーク，関係を含む図式モデルの全ての要素を ELEMENT として抽象シグネチャで定義し，要素は NAME と ELEMENTTYPE を持つようにする．ELEMENT の種類を識別するために ELEMENTTYPE を抽象シグネチャとして定義する．ELEMENTTYPE は主にメタ階層制約の関係を実現するために用いる．

NODE と ARC は ELEMENT を継承してシグネチャとして定義する．NODE と ARC は図式モデルの各要素の種類であり，図式モデルを作成する場合に，作成する要素がこの NODE か ARC のどちらであるかを決定する必要がある．図式モデルの各要素は NODE または ARC を継承し定義する．

メタ階層制約に関しては，メタメタモデルにおけるアークの制約として，メタメタモデルで定義されている要素間の関係制約に従っているかを検証する述語を定義する．このとき，メタ階層のシグネチャをシングルトンシグネチャとして定義する．メタ階層はレベルの異なる図式モデル要素間で関

係を作るので，メタ階層制約の関係は図式モデル上でグローバルに存在しなければならない．よって，メタ階層制約の関係をオーバーライド演算で追加していく．

図4にあるメタメタモデルはENTITY，RELATION，FIELD，CONTAINの4つであり，これらの要素はそれぞれノード，アーク，ノード，アークを継承してシグネチャとして定義する．またENTITY，RELATION，FIELD，CONTAINの型もシグネチャとして定義する．

そして，メタメタレベルにおけるこれらの要素間の制約を管理するシグネチャCONNECTIONを持つシングルトンなシグネチャを定義する．

また，図4の関係制約を不変条件として記述する．不変条件とした理由は，メタメタモデルは固定であることが挙げられる．

最後に，メタメタモデルの初期化ではメタメタモデルのメタな要素を決定するための述語を定義する．また，メタメタモデルのメタモデルはメタメタモデルであるため，各要素は自身の要素とメタ関係を持つような記述にする必要がある．そして，メタメタモデルの要素は自身の型への関連を持つことで要素からメタ制約関係を満足できるようになる．

メタモデルの記述もメタメタモデルの要素と同様に，メタモデル要素と要素の型のシグネチャ，メタモデル制約を管理するシグネチャ，メタモデルの制約記述する述語，メタモデルにおけるアーク制約記述する述語，メタモデルの初期化を行う述語を記述する．

3 DFDの階層化による適用評価

段階的検証法の適用評価として，2章で述べたDFDメタモデルに階層化の概念を追加したときのAlloyにおける記述方法と検証および評価について述べていく．

3.1 DFDの階層化

DFDの階層化の概念は，最上位レベルはコンテキストダイアグラムとし，最上位以外の下位レベルは，ある1つのプロセスを見たときに，そのプロセスの詳細なデータフローを記述する．階層化における整合性は上位のあるプロセスの入出力データフローがそのプロセスの下位のレベルのデータフローにおける入出力プロセス群の種類と一致することである．階層化可能なDFDのメタモデルを図9，図10，図11に示す．

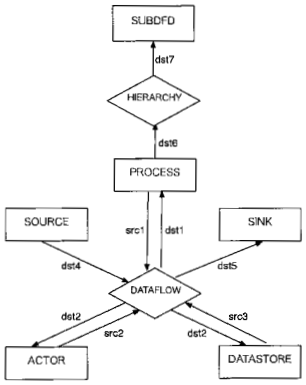

図 9: 階層化可能なDFDのメタモデル

これらのモデルは図6を拡張したものであり，追加要素としてSOURCE，SINK，SUBDFD，HIERARCHY，TOPDFD，OWNを追加する．図9のモデルは各要素間のリレーションシップ関係を図示している．SOURCEは階層化時の最上位でない下位レベルでの入力データフローを特定するために追加したエンティティであり，入力データフローの接続元のノードタイプのエンティティとして存在する．SINKは下位レベルでの出力フローを特定するために追加したエンティティであり，出力データフローの接続先のノードタイプのエンティティとして存在する．SUBDFDはあるプロセスを階層化したときの下位レベルのDFDを持つフィールドである．HIERARCHYはPROCESSとSUBDFDの関連として存在し，プロセスが下位のDFDを所有するという概念をベースモデル上で表現するために定義する．

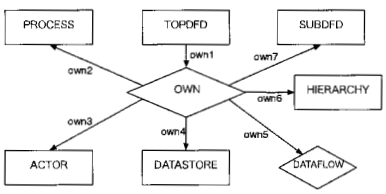

図 10: TOPDFDがCONTAINする要素

TOPDFDは最上位レベルのDFDを持つフィールドとして定義する．図10はTOPDFDが所有する要素を図示しており，TOPDFDはPROCESS，

ACTOR, DATASTORE, DATAFLOW, HIERARCHY, SUBDFD を所有する．OWN はメタメタモデルのコンテインとして定義する．TOPDFD では SUBDFD が所有する SINK，SOURCE へは所有しない．

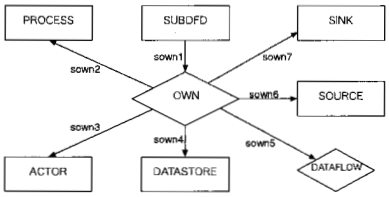

図 11: SUBDFD が CONTAIN する要素

SUBDFD は下位レベルの DFD を持つフィールドとして定義する．図 11 は SUBDFD が所有する要素を図示しており，SUBDFD では PROCESS, ACTOR, DATASTORE, DATAFLOW, SOURCE, SINK を所有する．

またメタメタモデルとのメタ制約関係は，SOURCE と SINK が ENTITY に対応し，HIERARCHY が RELATION に対応し，TOPDFD と SUBDFD が FIELD に対応し，OWN が CONTAIN に対応する．

このようにして，DFD メタモデルに階層化の概念を取り入れた．この DFD メタモデルを Alloy 上で記述し，検証を行なう．

3.2 記述方法

階層化のときに追加したメタモデル要素 SOURCE, SINK, TOPDFD, SUBDFD をノードタイプのエンティティとして定義し，HIERARCHY, OWN はアークタイプのエンティティとして定義する．またメタ制約関係を作成するためにこれらの型も定義する．そして，図 9，図 10，図 11 の関係制約を述語として記述し，階層化可能な DFD メタモデルにおけるアークの制約を記述する．階層化可能な DFD メタモデルの初期化を行う述語では，メタ制約関係を定義する．

3.3 検証方法

本節では Alloy の run コマンドで DFD のメタモデルからこのメタモデルの制約に従った DFD のベースモデルのインスタンスが存在するかを検証する．これによって，メタレベルにおける記述が正しいことが保証される．

検証を行なうスコープは，ある一つのプロセスに着目し，そのプロセスの階層化が正しく行われているかを検証する．これによって，任意のプロセスにおいて階層化が可能なことが保証される．

今回作成したメタモデルでは図式モデル要素の関連を追加定義しただけなので，図式モデルのセマンティクスを考慮した制約に関する記述を行なっていない．そこで，階層化可能な DFD の制約要件を定義しなければならない．階層化可能な DFD の制約要件は以下の通りである．

1. TOPDFD は 1 つしか存在しない
2. PROCESS には SUBDFD への関連が必ず一つだけある
3. SUBDFD は必ず SOURCE を一つだけ持つ
4. SUBDFD は必ず SINK を一つだけ持つ
5. TOPDFD は SOURCE を持たない
6. TOPDFD は SINK を持たない
7. SUBDFD には PROCESS が必ず存在する
8. TOPDFD には PROCESS が必ず存在する
9. SOURCE と SINK には必ず DATAFLOW が存在しなければならない．そして，DATAFLOW の接続先は PROCESS でなければならない
10. PROCESS の前後には必ず入出力の DATAFLOW が存在しなければならない
11. PROCESS, ACTOR, DATASTORE の出力 DATAFLOW が自 PROCESS の入力プロセスにはならない (自己参照の禁止)
12. TOPDFD が所有する要素に対して，出力 DATAFLOW がありかつ入力 DATAFLOW がない ACTOR と，入力 DATAFLOW がありかつ出力 DATAFLOW がない ACTOR が存在する (コンテキストダイアグラムの制約)
13. すべての要素は TOPDFD または SUBDFD が所有する要素である
14. 階層間のデータフローは存在しない
15. 階層化の整合性問題

上記の要件を満たす制約を Alloy 上で記述し検証する．記述は fact 文で記述する．評価するための検証方法としては要件 1 から要件 15 までを，論理積で順次演算していき，Alloy Analyzer を通して充足可能性問題で解を探索する．run コマンドは以下のような記述で行なった．なお，コマンド引数の n はスコープ数である．また，制約要件で用いた Alloy 記述を図 12 に示す．

```
fact restrictionSUBDFDMODEL{
(#TOPDFD=1)//要件 1
&&(one a1:HIERARCHY | a1.begin=PROCESS && a1.end=SUBDFD) //要件 2
&&(one a2:OWN | (a2.begin=SUBDFD && a2.end=SOURCE)) //要件 3
&&(one a3:OWN | (a3.begin=SUBDFD && a3.end=SINK)) //要件 4
&&(no a4:OWN | (a4.begin=TOPDFD && a4.end=SOURCE)) //要件 5
&&(no a5:OWN | (a5.begin=TOPDFD && a5.end=SINK)) //要件 6
&&(some a6: OWN | a6.begin=SUBDFD && a6.end=PROCESS) //要件 7
&&(some a7:OWN | (a7.begin=TOPDFD && a7.end=PROCESS)) //要件 8
&&((some d1: DATAFLOW | (d1.begin=SOURCE)&&(d1.end=PROCESS))&&(some d2:DATAFLOW | (d2.begin=PROCESS)&&(d2.end=SINK))) //要件 9
&&(some p1: PROCESS, d3,d4: DATAFLOW | (d3.end=p1)&&(d4.begin=p1)) //要件 10
&&(all d5: DATAFLOW | (no e2:ELEMENT | d5.begin=e2 && d5.end=e2)) //要件 11
&&(some o1,o2 :OWN, ac1, ac2 : ACTOR | o1.begin = TOPDFD && o2.begin= TOPDFD && o1.end = ac1 && o2.end = ac2 &&
(some d6, d7, d8, d9: DATAFLOW| d6.begin = ac1 && d7.end != ac1 && d8.end = ac2 && d9.begin != ac2)) //要件 12(コンテキストダイアグラムの制約)
&&(all p:PROCESS | (some a8: OWN|(a8.begin=TOPDFD || a8.begin=SUBDFD)&&(a8.end=p)))
&&(all df:DATAFLOW | (some a8: OWN|(a8.begin=TOPDFD || a8.begin=SUBDFD)&&(a8.end=df)))
&&(all a:ACTOR | (some a8: OWN|(a8.begin=TOPDFD || a8.begin=SUBDFD)&&(a8.end=a)))
&&(all ds:DATASTORE | (some a8: OWN|(a8.begin=TOPDFD || a8.begin=SUBDFD)&&(a8.end=ds))) //要件 13
 ...
}
```

図 12: 制約要件を記述する fact 文

```
run initSUBDFDMETAMODEL for n but
1 metameta/METAMETAMODEL, 1 SUBDFDMETAMODEL,
1 TOPDFD, 1 SUBDFD, 1 SOURCE, 1 SINK
```

3.4 検証結果

前節の要件を満たすモデルの探索を Alloy Analyzer の充足可能性問題で行なった．探索結果を表 1 に示す．なお solver は sat4j を用いた．

要件 1〜13 の検証時は，解は発見できず，探索時間の合計は 8238320(ms) だったため，本研究ではこれ以上の検証は実用性はないと判断し，実験を中止した．要件 1〜12 までは解が見つかったため，コンテキストダイアグラムの制約までは成功したといえる．しかし，結果として階層化の整合性を検証する段階までには到達できなかったともいえる．これは，要件 13 の制約記述で全称記号を用いた 4 つの式を論理積で演算するため，計算量が飛躍的に増加してしまうことが考えられる．

表 1: モデルの探索結果

要件	スコープ	要素数	探索時間
なし	14	4192	9855(ms)
1	14	4192	10249(ms)
1〜2	14	4192	9997(ms)
1〜3	14	4192	9727(ms)
1〜4	15	4521	11725(ms)
1〜5	15	4521	12296(ms)
1〜6	15	4521	12361(ms)
1〜7	16	4872	13697(ms)
1〜8	17	5231	17874(ms)
1〜9	18	5616	21721(ms)
1〜10	18	5670	21128(ms)
1〜11	18	5670	22208(ms)
1〜12	21	6936	34991(ms)
1〜13	-	-	-
1〜14	-	-	-
1〜15	-	-	-

3.5 検証結果の視覚化

AlloyVisualizer で表示された結果を図 13 に示す．Visualizer で視覚化が理解しやすいようにするために，アークがどのノードに接続しているかを中心に表示できるように設定した．図 13 から DATAFLOW0 の接続元 (begin) は PROCESS であり，接続先 (end) は ACTOR である．また，DATAFLOW1 の接続元 (begin) は ACTOR であり，接続先 (end) は PROCESS である．このグラフから要件 12 のコンテキストダイアグラムの制約を満たしていることが読み取れる．しかし，TOPDFD は PROCESS と ACTOR を所有しているが，DATAFLOW を所有していない．これを対策するには要件 13 の制約を満たさなければならないことがわかる．

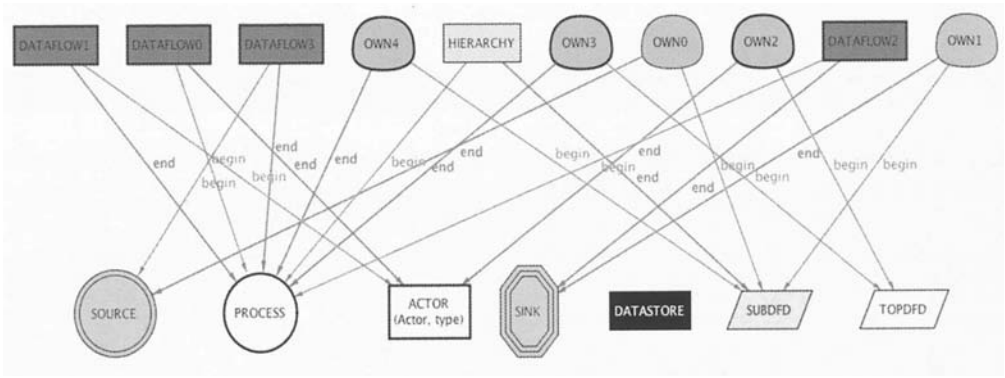

図 13: Visualizer で表示される関係論理グラフ (要件 1〜12)

4 まとめ

本研究ではメタ階層アーキテクチャを採用している図式モデルを対象に，Alloy を用いた段階的な検証方法を提案した．Alloy では検証コマンドが run コマンドと check コマンドだけなので，これの切り分けを決定することで，検証すべき対象を明確にすることができる．check コマンドの場合は上のレベルのモデルの制約を満たしているかどうかの検証を使うことができる．また妥当性の確認という意味で，run コマンドでは主に下のレベルのモデルの構築例を見ることにより，現在のレベルのモデルの妥当性を確認できる．この手法を用いることで，例えばこのようなメタ階層を持っているモデルの場合，メタモデルが変更されたらそれに応じてベースモデルも変更されなければならない．このときベースモデルの不整合等の検出も容易になると考えられる．また新たなメタモデルを構築する際に，ベースモデルが作成者の意図に沿ったものかどうか，制約がどれくらい足りないかといった利用者の認識不足の解消にも役立てることができるものと考えられる．さらに実際に Alloy 上で自動検証が行なえるよう，図式モデルを Alloy 記述で記述できる方法も示した．今回の方法では，モデルの作成は利用者が自ら記述していかなければならないが，本研究のような Alloy 記述に対応可能なモデルを多数用意できれば，形式手法の支援にもつながっていくものと考えられる．

今後の課題としては，今回 DFD の階層化を試み，ベースモデルの生成を試みたが，階層化における整合性検証に到達するまでには至らなかった．これは，AlloyAnalyzer の充足可能性問題を解くのに膨大な時間がかかってしまったことが挙げられる．原因としては DFD メタモデルの制約記述が複雑あり，今回 DFD に記述した制約式を再考する必要がある．また図式モデルの要素間の関係は，今回アーク制約を中心に記述していったので，ノードの方にも制約を記述すれば改善できるのではないかと予想される．また，今回 check コマンドの詳細な検証方法には言及しなかったので，check コマンドの詳細な検証方法について考えていく必要がある．

参考文献

[1] 中島 震：ソフトウェア工学の道具としての形式手法，NII-2007-007J (2007).

[2] J.M. Spivey: The Z notation: A Reference Manual - Second Edition - ,Prentice-Hall (1992).

[3] Object-Z Tool Support, http://www.itee.uq.edu.au/smith/tools.html.

[4] 小飼 敬，上田 賀一：メタ階層化による図式モデルの形式的記述，情報処理学会研究報告，Vol.2001, No.56, pp. 41-48.

[5] ZANS TOOL, http://venus.cs.depaul.edu/fm/zans.html

[6] Alloy, http://alloy.mit.edu/