ソフトウェア工学 77−11
（1991. 2. 7）

汎用的なダイヤグラム法表現モデルとエディタ

新田稔
(株) SRA
ソフトウェア工学研究所

鳥居宏次
大阪大学
基礎工学部

ソフトウェアの設計や分析のための種々多彩なダイヤグラム法に対して多くの支援ツールが存在するが、特定のダイヤグラム法を固く組み込んでいるために、柔軟性に欠けるツールが多い。我々は、ユーザが自由にダイヤグラム法を新規考案・部分変更できるよう、ダイヤグラム法を一般的に表現するモデルを設定し、そのモデルによるダイヤグラム法のルール(定義)を取り込んで目的のダイヤグラム法専用のエディタとなるツールを開発している。ルール自体もまたメタ・ルールとして定義されたダイヤグラム法によって作成される。ルールに含まれるのは、ダイヤグラムの要素の型の定義、要素の組合せ方、ダイヤグラムの動的な振舞いなどである。

A General Model and a Meta-Editor for Diagramming Methods

Minoru Nitta
Software Research Associates,Inc
Software Engineering Laboratory

Koji Torii
Osaka University
Faculty of Engineering Science

There are many support tools of various diagramming methods for software design and analysis. But most of them are not so flexible because specific methods are tightly built in them. To allow users modifying the method of a tool freely, we have emploied a generic model of the diagramming methods and are developing a diagram editor which becomes to support a specific method when a rule (definition) of the method is given. Rules themselves will be generated based on the "Meta-rule" diagramming method. The rule consists; types of the diagram elements, combination of elements and dynamic behavior of the diagram.

はじめに

ソフトウェアの分析や設計にダイヤグラムが用いられることが多い。ダイヤグラムは、ひとりの人間の作業では思考を明確にするための道具であり、また複数の人間が共同作業するときには意志疎通の道具にもなる。現在、種々多彩なダイヤグラムを用いた分析や設計の方法(これらを総括的に「ダイヤグラム法」と呼ぶ[1])に対して、それらに従った作業をコンピュータによって支援するツール(CASEツール[2])が開発されている。しかし従来のツールの多くは、支援するダイヤグラム法を固く組み込んでいるために、いくつかの点において柔軟性に欠けていた。我々は

- ●新たに考案されたダイヤグラム法に対し、支援ツールを簡単に素早く構築する
- ●開発現場の状況に合わせて、ダイヤグラム法の細かな点がチューニング可能である
- ●異なるダイヤグラム法の支援ツール間でデータが交換できる

などのためには、ツールとそれが支援するダイヤグラム法との独立が必要であると考え、種々のダイヤグラム法を統一的に表現するモデルGDMを設定し、このモデルによるダイヤグラム法のルールを読み込むとそのダイヤグラム法の専用エディタとなるツールSERA-DEを開発している[3]。GDMではダイヤグラム法のルール自体もまた、ひとつのダイヤグラムとして表現可能であり、そのルールを作成するダイヤグラム法のルールを「メタ・ルール」と呼ぶ。本稿は、このGDMとメタ・ルールによるダイヤグラム法のルールの定義方法、およびルールの例を紹介するものである。

1.ダイヤグラム法表現モデル(GDM)

GDMでは、ダイヤグラムの構成要素を、

【ノード】独立に存在し意味を持つ要素。アイコンで表示される。

【エンクロージャ】ノードや他のエンクロージャを囲む要素。矩形の枠で表示される。

【アーク】二つのノード、エンクロージャ、またはアークを結ぶ要素。始点・終点に矢印などのシンボルを持つ線分(実線や点線などのスタイル)で表示される。上方から下方へのみ、または左方から右方へのみという方向制限が付けられることもある。

【プロパティ】ノード、エンクロージャ、およびアークを修飾する要素。テキストで示される(常に表示されるか、あるいは要求に応じて表示される)。値を持つものと持たないものがある。値には整数や実数、文字列などの型があり、取り得る値の最大値・最小値または選択枝や、値がユニークでなければならない範囲を定めることもできる。値を持たないプロパティは、修飾すること/しないこと自体が真偽値を示す。

の四種類に分類し、個々のダイヤグラム法によってこれらの要素にいくつかの【型】が設定されるものとしている。SERA-DEは特定の型の要素の組合せに誤りが起きないよう、メニュー項目選択可/不可のコントロール、リアルタイムな誤り部分の指摘、およびユーザの指示によるダイヤグラム全体の検査によって、ダイヤグラム作成操作をガイドする。

GDMでは、(部分的な)ダイヤグラムを入れる容器を【レイヤ】と呼ぶ。レイヤは、ノードがひとつのレイヤを所有するという関係によって、ダイヤグラム全体を木構造に整理する。ただし最上層のレイヤ(ルート・レイヤ)は、どのノードにも所有されない。SERA-DEはそれぞれのレイヤをひとつのウィンドウで表現する。

ダイヤグラム法は、単にダイヤグラムを作成・分析する技法だけでなく、ダイヤグラムに基づく他のドキュメントの生成や記述対象のシミュレーション、プロセスの起動などのような動的な機能を含んでいる。GDMではこのようなダイヤグラムの動的な振舞いを、ダイヤグラムの要素に【メッセージ】を送るとそれに対応する【スクリプト】が実行されるものとしている。SERA-DEでは、ユーザが該当の要素を選択しメニューからメッセージを選ぶことでこれを行なう。またスクリプト内から他の要素へメッセージを送ることもできる。

2.ダイヤグラム法のルール定義

2.1.要素の型の定義

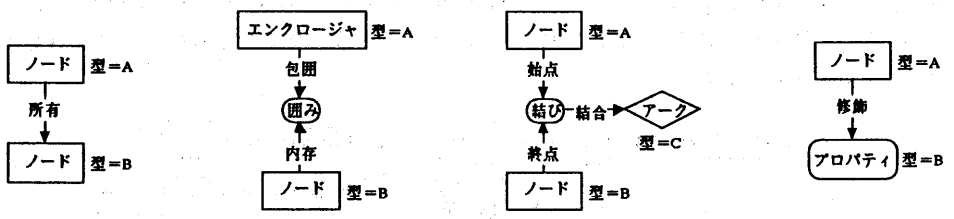
メタ・ルールでは、目的のダイヤグラム法が持つノードやエンクロージャ、アーク、プロパティの型を、〈ノード〉型、〈エンクロージャ〉型、〈アーク〉型、または〈プロパティ〉型のノードを作成し、それを〈型〉型プロパティで修飾することによって定義する。〈型〉型プロパティの値が定義しようとする型を示す。これらのノードはこの他にもいくつかのプロパティによって修飾され、定義する型の要素の性質が規定される(たとえば〈ノード〉型ノードは、その型のノードの図形表現を指定する〈アイコン〉型、その型のノードが必ずレイヤを所有しなければならないことを示す〈レイヤ必須〉型、ルート・レイヤには存在できないことを示す〈ルート・レイヤ存在不可〉型のプロパティに修飾される)。

2.2.組合せの定義

要素は次の四つの方式で組み合わされる。

【所有組合せ】ある型のノードが所有するレイヤに、ある型のノードやエンクロージャが存在するという組合せである。たとえば『(A)型のノードが所有するレイヤに(B)型のノードが存在する』を定義するには、〈ノード〉型ノード(A)から〈ノード〉型ノード(B)へ〈所有〉型アーク引く(図1-a)。〈所有〉型アークは、〈組合せ詳細〉型プロパティによって修飾されることがある。その値は「少なくともひとつ」「必ずひとつだけ」「多くともひとつ」のいずれかであり、ひとつのレイヤに存在するその型のノードやエンクロージャの数の制限を示す。修飾されなければ制限はない。

【包囲・内在組合せ】ある型のエンクロージャが、ある型のノードやエンクロージャを囲むという組合せである。たとえば『(A)型のエンクロージャが(B)型のノードを囲む』を定義するには、〈囲み〉型ノードを作成し、そこへ〈エンクロージャ〉型ノード(A)から〈包囲〉型アークを、〈ノード〉型ノード(B)から〈内存〉型アークを引く(図1-b)。〈包囲〉型と〈内存〉型のアークは、〈組合せ詳細〉型プロパティによって修飾されることがあり、それぞれ、ひとつのエンクロージャがその型のノードやエンクロージャを囲む数の制限、ひとつのノードやエンクロージャがその型のエンクロージャに囲まれる数の制限を示す。



a 所有・存在組合せ　b 包囲・内在組合せ　c 結合・始点・終点組合せ　d 修飾組合せ

図1 要素の組合せの定義

【結合・始点・終点組合せ】ある型のアークが、ある型のノードやエンクロージャ、アークを始点・終点として結ぶという組合せである。たとえば『(C)型のアークが(A)型のノードを始点とし(B)型のノードを終点として結ぶ』を定義するには、まず〈結び〉型ノードを作成し、そこへ〈ノード〉型ノード(A)から〈始点〉型アークを、〈ノード〉型ノード(B)から〈終点〉型アークを引いて始点と終点の要素の型を示す。次に〈結び〉型ノードから〈アーク〉型ノード(C)へ〈結合〉型アークを引く(図1-c)。〈始点〉型と〈終点〉型のアークは、〈組合せ詳細〉型プロパティによって修飾されることが

あり、それぞれ、ひとつのノードやエンクロージャ、アークがその型のアークの始点または終点となる数の制限を示す。

【修飾組合せ】ある型のプロパティが、ある型のノードやエンクロージャ、アークを修飾するという組合せである。たとえば『(B)型のプロパティが(A)型のノードを修飾する』を定義するには、〈ノード〉型ノード(A)から〈プロパティ〉型ノード(B)へ〈修飾〉型アークを引く(図1-d)。〈修飾〉型アークは〈組合せ詳細〉型プロパティによって修飾されることがあり、これはひとつのノードやエンクロージャ、アークがその型のプロパティによって修飾される数の制限を示す。

またGDMでは、このような要素の組合せの二つ以上の組に対して、

- ●少なくともどれかひとつは発生しなければならない
- ●必ずひとつだけ発生しなければならない
- ●多くともどれかひとつだけしか発生してはならない
- ●ひとつが発生すると他も発生しなければならない
- ●発生した数が対応していなければならない

の制限を設けることができる。メタ・ルールではこれらを、〈組合せ制限〉型ノードと〈制限詳細〉型プロパティを用いて定義する。その一般形は、ひとつの〈組合せ制限〉型ノードから制限の対象となる組合わせを示す〈所有〉型、〈包囲〉型、〈内存〉型、〈始点〉型、〈終点〉型、あるいは〈修飾〉型のアークへ〈要請〉型アークを引き、〈組合せ制限〉型ノードを〈制限詳細〉型プロパティによって修飾するものである。〈制限詳細〉型プロパティの値は、「少なくともひとつ」「必ずひとつだけ」「多くともひとつ」「同時に発生」「対応して発生」のいずれかで、制限の内容を示している。さらにGDMでは、この組合せ間の制限が、ある条件が成り立っているとき(他の組合せがあるとき)のみのものであるとすることができる。条件としては、

- ●ノードが所有するレイヤにある型のノードまたはエンクロージャが存在している
- ●エンクロージャがある型のノードまたはエンクロージャを囲んでいる
- ●ノードまたはエンクロージャがある型のエンクロージャに囲まれている
- ●ノード、エンクロージャ、またはアークがある型のアークの始点・終点になっている
- ●ノード、エンクロージャ、またはアークがある型のプロパティで修飾されている

がある。メタ・ルールではこれらを、条件となる組合わせを示す〈所有〉型、〈包囲〉型、〈内存〉型、〈始点〉型、〈終点〉型、あるいは〈修飾〉型アークから組合わせの制限を示す〈組合せ制限〉型ノードへ〈要請〉型アークを引いて示す。たとえば図2のルールでは『(A)型のエンクロージャは、(D)型のプロパティに修飾されるときには、(B)型か(C)型のノードを少なくともひとつ囲まなければならない』が定義されている。

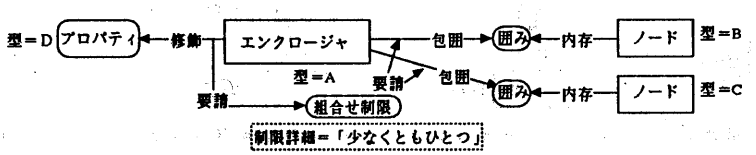


図2 組合せ間の制限

2.3.メッセージの定義

メタ・ルールでは、ある型のノード、エンクロージャ、アーク、プロパティ、またはレイヤ全体に送るメッセージを、〈メッセージ〉型ノード、〈送信〉型アーク、〈スクリプト〉型と〈呼出し方式〉型のプロパティを用いて定義する。たとえば『(A)型のノードへ(B)型のメッセージが送られる』を定義するには、〈メッセージ〉型ノード(B)から〈ノード〉型ノード(A)へ〈発信〉型アークを引く。レイヤ全体へ送るメッセージの〈メッセージ〉型ノードからはどこへも〈発信〉型アークを引かない。

〈メッセージ〉型ノードは〈スクリプト〉型と〈呼出し方式〉型のプロパティによって修飾される。〈スクリプト〉型プロパティの値は一種のプログラムであり、関数型プログラミング言語akanc[4]のサブセットに、要素の検索や値の設定、表示操作(ハイライト、網かけ、ブリンク)、プロセス起動やファイル入出力のシステム・コールなどを行なう組込み関数を追加した言語によって記述される。また〈呼出し方式〉型プロパティの値は、メッセージがユーザの操作によってのみ発信されることを示す「直接呼出し」か、あるいは他のスクリプトからのみ発信されることを示す「間接呼出し」のいずれかである。〈呼出し方式〉型プロパティによって修飾されないときは、どちらの方式でも呼び出されることを示す。

3.ダイヤグラム法のルールの例

ダイヤグラム法のルールの例としてペトリネットのルールを挙げる。このルールは、ペトリネット・グラフに現れる「プレース」「イベント」「アーク」という要素の型やその組合せ方式と、「イベント」に「発火」というメッセージを送ったときのグラフの振舞いを定義している(図3)。

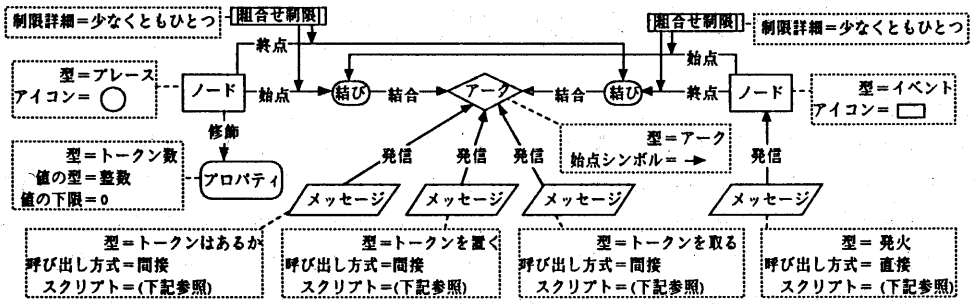


図3 ペトリネットのルール

```
/* 「発火」の〈スクリプト〉型プロパティの値 (下線は組込み関数を示す) */
bool check(neap);              /* neap型の引数をひとつ取るcheckの型はbool型 */
state get_token(neap,state); /* neap型とstate型の引数を取るget_tokenの型はstate型 */
state put_token(neap,state); /* neap型とstate型の引数を取るput_tokenの型はstate型 */

state main(neap event,state s)
==    if check(event,s)                              /* イベントが発火できる? */
      then  put_token(event,get_token(event,s))      /* トークンを移す */
      else  s ;                                      /* なにもしない */

bool check(neap event)
==    for arc=>comming_arc(event)          /* イベントを終点とするアークについて */
      do    if arc = null                  /* 全てのアークを調べ終った? */
            then true                      /* イベントは発火可能 */
            else if send(arc,"トークンはあるか",s,(bool))
                                           /* アークへ"トークンがあるか"を送る */
            then  next(comming_arc(event,arc))
                                           /* 次のアークを調べる */
            else  false ;                  /* 発火は不可能 */

state get_token(neap event,state s)
==    state for arc=>comming_arc(event),s=>s  /* イベントへ来ているアークについて */
      do    if arc = null                     /* 全てのアークについて処理が終った? */
            then  s                           /* そのままの状態を返す */
            else  next(comming_arc(evenet,arc),send(arc,"トークンを取る",s,(state))) ;
                  /* アークへ"トークンを取る"。その後の状態で次のアークを処理 */

state put_token(neap event,state s)
```

```
==    state for arc=>leaving_arc(event),s=>s  /* イベントから出ているアークについて */
      do    if arc = null                    /* 全てのアークについて処理が終った? */
            then s                           /* そのままの状態を返す */
            else next(leaving_arc(evenet,arc),send(arc,"トークンを置く",s,(state))) ;
                 /* アークへ"トークンを置く"。 その後の状態で次のアークを処理 */

/* 「トークンはあるか」の<スクリプト>型プロパティの値 */
bool main(neap arc,state s) == value(property(origin(arc),"トークン数"),(int)) > 0 ;
                 /* アークがの入力プレースのトークン数はゼロ以上? */

/* 「トークンを取る」の<スクリプト>型プロパティの値 */
state main(neap arc,state s)
with  PLACE # origin(arc),                   /* 入力プレースをPLACEとする */
      TOKEN # property(PLACE,"トークン数"),  /* そのトークン数をTOKENとする */
==    blink(arc,set(TOKEN,value(TOKEN,(int)) - 1,blink(PLACE,s))) ;
/* 入力プレースをブリンク、トークン数に1小さい値をセット、アークをブリンク */

/* 「トークンを置く」の<スクリプト>型プロパティの値 */
state main(neap arc,state s)
with  PLACE # destination(arc),              /* 出力プレースをPLACEとする */
      TOKEN # property(PLACE,"トークン数"),  /* そのトークン数をTOKENとする */
==    blink(PLACE,set(TOKEN,value(TOKEN,(int)) + 1,blink(arc,s))) ;
/* アークをブリンク、トークン数に1大きい値をセット、出力プレースをブリンク */
```

おわりに

ダイヤグラムの要素の型の定義と組合せの定義の方法については、過去に類時のツール[5]の開発経験があり、かなり整理され完成度の高いものになっている。しかしダイヤグラムの動的な振舞いの定義方法は、以前のツール[6]ではプログラム・コードとして組み込まれていたものを、ユーザが直接さわることのできる形に取り出したものである。関数型言語の採用は、静的検査が手続き型言語よりも厳密に行なえることによる実行時エラーの減少を期待してのことである。この定義方法についてはまだ実績はなく、記法も未消化で記述が易しいとは言えない。今後の課題として、多くのダイヤグラム法のルールの定義を蓄積してダイヤグラムの動的な振舞いをパターン化し、より効率的な定義方法を設定したい。

なお本稿で紹介できたルールの定義方法はメタ・ルールの一部分である。メタ・ルール全体の詳細については[7]を参照していただきたい。またSERA-DEは、共同作業支援機構SERA中のひとつのツールであり、ネットワークでつながれたワークステーションから複数のユーザが同時にひとつのダイヤグラムについて作業できることや、異なったルールを取り込んだSERA-DE、および他のツールとも協調して動作することを付け加えておく。

参照資料

[1] J.Martin, C.McClure "Diagramming Techniques for Analysts and Programmers" Prentice-Hall 1985
「ソフトウェア構造化技法 ダイヤグラム法による」近代科学社（國友、渡辺 訳）

[2] 日経コンピュータ「ソフト開発の上流工程を支援するCASEツール」1989年2月13日号

[3] 歌代、石塚、新田「SERA ダイヤグラム法を用いた多人数による分析/設計作業支援ツール」第16回 jus UNIXシンポジウム(大阪) 1990

[4] 新田、鳥居「akanc 大規模ソフトウェアのための関数型プログラミング言語」情報処理学会研究会報告 90-SE-73 1990

[5] 新田「ERAS モデリング・エディタ」 (株)ＳＲＡ 技術資料 1987

[6] 新田「MOSS モデリング・エディタのアニメーション・エンジン」(株)ＳＲＡ 技術資料 1987

[7] 新田「メタ・ルールによるダイヤグラム法の定義」 (株)ＳＲＡ 技術資料 1990