グラフィカル・メタモデリング
(Graphical GOPRR) 概説書

Version 4.5
The Graphical Metamodeling Example

MetaCase Document No. GE-4.5

First Printing, 2nd Edition, February 2008.

MetaCase
Ylistönmäentie 31
FI–40500 Jyväskylä
Finland
Tel: +358 14 4451 400
Fax: +358 14 4451 405
E-mail: info@metacase.com
WWW: http://www.metacase.com

はじめに

この資料ではサンプルを用いて、MetaEdit+でグラフィカルにメタモデリングできることを紹介します。MetaEdit+のグラフィカル・メタモデリングによりメタモデルの主となる、言語コンセプトと関連するルールといった基本部分をデザインすることで、言語（モデリング環境）作成の始めの段階を支援します。作成された言語のデザインは、MetaEdit+ Workbenchにインポートされ、そのメタモデリングツール（機能）を用いて、拡張（進化）させられます。

理論的に言えば、メタモデリング言語はMetaEdit+ Workbenchにある多くのドメイン・スペシフィック言語の一つです。このドメインとは、モデリング言語をデザインし、言語の定義を生成させてMetaEdit+で実行（メタモデルの拡張）をすることです。

この例では、メタモデリング言語を用いて作成される言語の定義から、XML形式のファイルを生成させます。このXMLファイルはMetaEdit+ Workbenchにモデリング言語としてインポートされます。以下、理解を深めるために、演習を行いながらの紹介となっています。その為には、MetaEdit+のインストールをされることをお勧めします。生成される言語を拡張（進化）させる場合には（表記シンボルの追加、追加のコンストレインツ・制限事項、コード生成機能、ダイヤログやツールバーの変更など）、MetaEdit+ Workbench　あるいはその評価版を使用下さい（www.metacase.com　からダウンロード可能です）

MetaEdit+　に関する詳しくは、‘MetaEdit+ Users Guide’、‘MetaEdit+ Workbench Users Guide’などインストール内のマニュアルやホームページを参考下さい
（http://www.fuji-setsu.co.jp/products/MetaEdit/index.html　）

1 グラフィカル・メタモデリングの例

グラフィカル・メタモデリングの例として、メタモデリング言語と、モデリング言語をデザインするツールサポートについて紹介します。　そして、言語コンセプト、プロパティ、接続、ルール、複数言語の統合など、メタモデルの基本にフォーカスします。

この章では、グラフィカル・メタモデリング言語と、その用法について。2章では、小さなサンプルを題材にユースケース図を作り出すことで、メタモデリング言語をどのように使用するかの説明。3章では、作られた言語仕様を変換し（Generate）、MetaEdit+ Workbenchへインポートし、確認・検証・拡張させる方法について。

このメタモデリング例から、メタモデリングのための言語について説明します。メタモデリング言語も、MetaEdit+内の他のメタモデル同様に実装されるので、MetaEdit+ Workbenchで拡張・進化させられます。しかしながら、その部分に関しては、この資料では触れません。

MetaEdit+によるメタモデリングのコンセプト、MetaEdit+使用法の基本、などは予め知識として必要です。評価概説資料　Family Tree example　が、これら知識を得るのに適しています。

1.1 グラフィカル・メタモデリングの基本

一般にグラフィカル・メタモデルは、プロパティ、接続、ルールなど基本的なモデリングコンセプトをカバーしています。MetaEdit+では、GOPRRというメタモデリングのコンセプト、graphs, objects, properties, relationships、roles を持っています。 グラフィカル・メタモデリングは、言語を作り出す初期の段階で、言語の基本構造をデザインし、関係者と意思疎通を図る上で役立てられます。グラフィカルなメタモデルも、言語の全体像を掴む上で有益です。

それは言語開発の後半になってくると、メタモデルだけではなく、それを用いたモデル例を参照できることで、メタモデルの作成（拡張など）が容易になるということです。
実践的なモデル例があれば、その言語をテストすることや、その言語のユーザに対しての理解を容易に出来るからです。
それゆえ、言語デザインの初期の段階をグラフィカルに行った後、MetaEdit+ Workbench のメタモデリング機能を用いて、言語を完成させることをお勧めします。
これはXMLベースのモデリング言語を、MetaEdit+ Workbenchにインポートすることで行えます。
そして、MetaEdit+ Workbenchで、シンブルの表記を加えたり、コンストレインツ、コードジェネレータ、などを定義し、ダイアログやツールバーを拡張したりすることで、言語を拡張します。

ここで留意することは、グラフィカル・メタモデリングは、言語を進化させることには向いていないということです。作り出そうとしているメタモデルと、メタモデルを作る為のメタモデルは異なりますので、実際のサンプルモデルを書いてテストすることなど出来ないからです。また、シンボルの表記、コードジェネレータ、ダイアログやツールバーへの修正はできません。これらの作業は、MetaEdit+ Workbench のシンボルエディタ、ジェネレータエディタ、ダイアログエディタなどを用いて、言語のデザイン、実装を深めることができます。

1.2 例となるメタモデル

グラフィカル・メタモデリング言語を用いて、ドメインスペシフィック言語を作り出すまでに、3つのステップに分解されます。始めにグラフィカル・メタモデリング言語でベーシックなコンセプトとルールを定義して、そのメタモデルをXML形式に変換（Generate）します。このファイルを言語定義としてMetaEdit+にインポートし、適用します。XMLへの変換（Generate）と、インポートについては3章で説明します。

Figure 1-1では、グラフィカル・メタモデリング言語でサンプルの言語仕様を図解しています。データフローダイアグラムを規定しています。データフローダイアグラムは、3つの基本エレメントである、‘External’, ‘Store’、‘Process’から構成されます。これらはメタモデリング言語のObjectのコンセプトを用いて定義されます。これら3つのオブジェクトは共通のプロパティ（properties）、データフロー内の似通った接続をもつので、‘Abstract’というオブジェクトタイプが言語に追加されています。これは、他のコンセプトの上位タイプ（super type）です。‘Abstract’オブジェクトは、‘{0}’によって、言語内の抽象コンセプト（abstract concept）とされます。

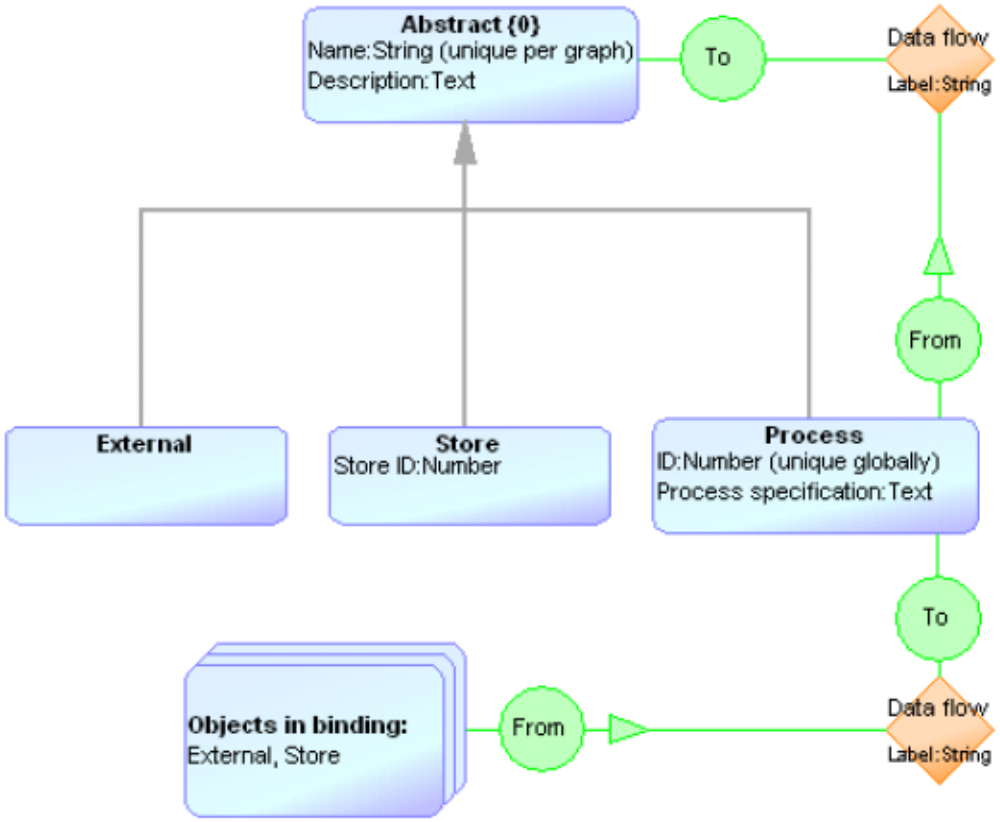


Figure 1-1. Metamodel of Data Flow Diagram

Figure 1-1のメタモデルには、モデリングのエレメントであるプロパティも見えています。例えば、‘Abstract’には2つのプロパティ、‘Name’（データフローダイアグラム内のユニーク値を持つ文字列）、‘Description’（データタイプ（テキスト））　があります。‘Name’がユニークであるというルールでは、同一データフローダイアグラム内に、同じ名前のStore, External、Processのインスタンスが存在できないことを規定しています。

他のモデルエレメントでは、さらなるプロパティを持っています。例えば、‘Process’は、プロパティタイプ　‘ID’を持っています。このプロパティは、数値データタイプであり、この値は全データフローダイアグラム間でユニークであること、‘globally’というキーワードで定義されるように、同じIDで2つの異なるプロセスが、いかなるダイアグラムにあってはなりません。

更にメタモデルを見ると、モデリング言語にオブジェクト間で2つのコネクションタイプがあります。これらをMetaEdit+では、バインディングと呼んでいます。
1つは、‘Process’から‘Abstract’のサブタイプへ。2つ目は、‘External’、あるいは‘Store’から‘Process’へ。2つ目のほうでは、‘Object set’というバインディング内のオブジェクト内容を表現する、メタモデリング言語のコンセプトを用いています。
これにより、個々のオブジェクトごとに、別個にバインディングを規定する必要が無くなり、メタモデルの作図が簡素化されます。

最後にバインディング内のエレメント、すなわちリレーションシップとロール、もプロパティをもつことができます。この例では、リレーションシップである‘Data Flow’は、ダイアグラム上2つ存在していますがただ一度だけ定義され、フローにラベルを入力するための文字列のプロパティタイプを持ちます。モデリング言語は他にも多くのルールを持ちますが、それらは後程紹介します。

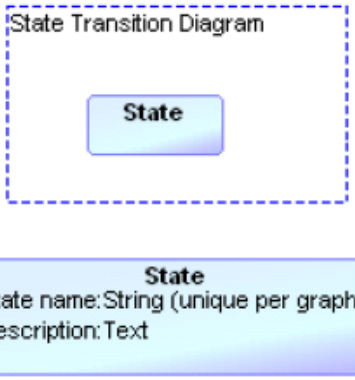
1.3 GOPRR メタモデリング言語について

グラフィカル・メタモデリング言語は、MetaEdit+’のGOPRRデータモデルをサポートする目的で作られました。それゆえ、この全てのモデリングコンセプトは、直接 MetaEdit+のものを用いています。これらメタモデリングのコンセプトは、

言語のコンセプト

Graph は、1つのモデル言語を規定。例えば、ステートダイアグラムやユースケースダイアグラムなど。各言語は、別個のメタモデルでモデル化される。言語同士の統合は、explosions とdecompositionsでメタモデル内に定義され、複数のグラフタイプをサポートする。.

Representation of the concept



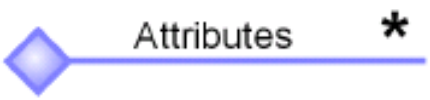
Object は、モデリング言語の基本コンセプトを記述する。デザインのメイン・エレメントとなる。 **Object** 同士は接続され、プロセス、メッセージ、ボタン、ステートなどのように、再利用され得る。

Relationship は、**Object** 同士の接続を定義する。インヘリタンス（継承）、メッセージ、コール、トランジションなど。**Object** や **Role** とバインディングを形成するために用いられる。

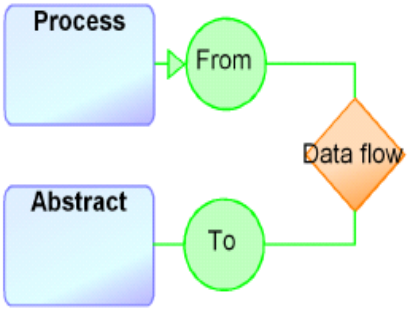
Role は、リレーションシップのライン（接続線）と終端を規定。継承リレーションシップのSuperclassや、状態遷移のFromなど。

Property は、これまでに紹介した言語コンセプトを特徴付けるアトリビュートを定義する。(string, text, number, Boolean, collectionなど異なるデータタイプで、他のモデリング言語のコンセプトや、ファイル/プログラム/ウエブサービスなどの外部ソースとリンクさせられる。**Property** の例は、ステート名、ファンクション識別子、ディスプレイタイプ、データタイプなど。

Property は、ほかの言語コンセプトの一部として表現される。例えば右図のように。



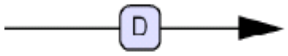
Bindingは、Relationship、2つ以上のRole、そして各Roleにはグラフ内の1つ以上のObjectsが接続される。Binding は、複数用いて階層に渡る設定もできる。



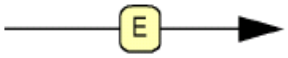
Object Setは、Binding 内で同じRoleとしての役割を果たしている Objectのコレクション（集合）。例えば、External とStoreは、同じデータフローRelationship内のFrom Role に存在する。

Inheritanceにより、他の言語コンセプトのサブタイプの生成を可能にする。例えば、Externalは、Abstractのサブタイプ。

Decompositionは、Objectsにサブグラフを持たせることができる。例えば、Process は、他のデータフローダイアグラムに decompose（デコンポーズ）できる。



Explosionは、Objects, Relationships, Roles から他のGraphsへのリンクを可能にする。例えば、データフローダイアグラム内のStoreの詳細構造が、Entity Relationship Diagramで規定されているなど。



グラフィカル・メタモデルをMetaEdit+のGOPRRへインポートさせる為に、このメタモデリングサンプルには、MXTファイルを生成するジェネレータを用意しています。生成されるMXTファイルは、MetaEdit+にインポートされ、モデリング言語として用いることができるようになります。MXTフォーマットに関して詳しくは、MetaEdit+ Workbench User's Guide を参考下さい。

2 メタモデリング言語の使用

この章では、グラフィカル・メタモデリング言語の起動や使用法を、既存のメタモデルや、新しいモデル言語（メタモデル）を作成することで紹介します。

2.1 メタモデルのサンプル例にアクセス

メタモデリングのサンプルにアクセスするには、MetaEdit+を起動し、GOPRRプロジェクトをdemoリポジトリから選択してログインします。そして、Graph Browserや Diagram Editorを用いてサンプルにアクセスします。

2.2 メタモデリング言語を触ってみる

メタモデリングのサンプルを評価するために、Graph Browser内にリストされるグラフを開きます。‘Structured Analysis and Design’という名前のグラフは、複数言語が統合されています。ダブルクリックして、この統合されたメタモデルを開きます、これは、以下のFigure 2-1と同じものです。

ダイアグラムには複数のグラフタイプがあり、これら4つのモデリング言語が、‘Structured Analysis and Design’として統合されている全体像を提供しています。ダイアグラム内で、各モデリング言語は大きな点線の四角形内で定義されています。 これはMetaEdit+のグラフタイプを意味します。各言語のコンセプトはグラフタイプシンボル内に図解されます。もし言語のエレメントが言語の一部として（グラフタイプシンボル内に）アタッチされなければ、エラーのテキスト表示が言語のエレメントに表示されます。
モデルエレメントのプロパティにアクセスするにはダイアグラム内で、あるいはDiagram Editorのサイドバーをダブルクリックします。各モデルアイテムに関連する動作に対しても、最初にエレメントを選択して、そのポップアップメニューを開くことでアクセスできます。

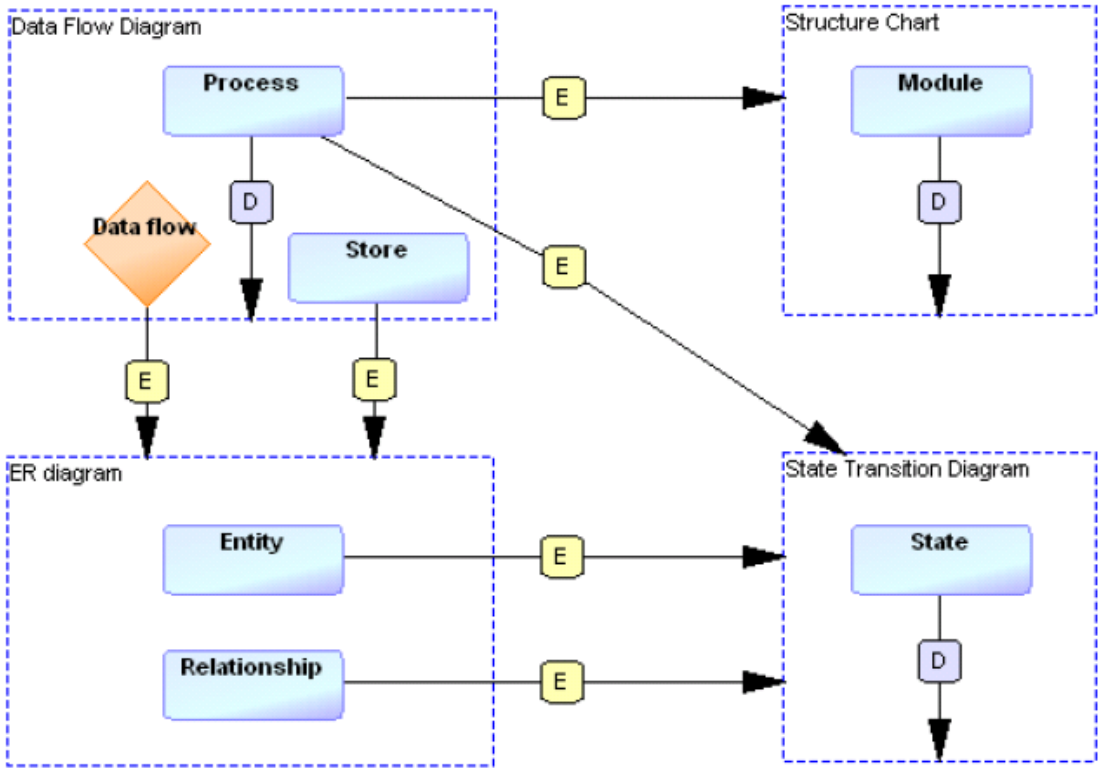


Figure 2-1. Metamodel for multiple graph types

他の言語に対するコネクションを持つ言語のエレメントは、relationshipでリンクされます。‘D’は、デコンポジション（decomposition）、‘E’ はエクスプロージョン（explosion）。Structured Analysis and Designメソッドに基づいて、Data Flow Diagram (Graph)内の‘Process’ (Object)は、Data Flow Diagramの他のインスタンス内にデコンポーズされます。‘Process’はまた、1つ以上のStructure Charts内にエクスプ

ロードされ、内部構造を規定します。また、State Transition Diagrams内にエクスプロードされ、振舞いを規定します。同様に、Data Flow Diagramにみられる‘Store’の図は、Entity-Relationship (ER) Diagramを用いて記述されます。他の言語コンセプト内の統合も、同様に図式化されています。

各グラフタイプの詳細は、別のメタモデリング言語で指定されています。これらの中身は、別のダイアグラムを開くことで確認できます。どれかグラフタイプ（点線四角形内のテキストシンボル）を、Ctrlキーを押しながらダブルクリックで開きます。

あるいは、ダイアグラム内でグラフタイプを選択し、左マウスクリックでポップアップメニューから**Decomposition**を選択することでも、グラフタイプの詳細ダイアグラムを開くことが出来ます。このポップアップメニューからは、decompositionのリンクを、現在のメタモデルから変更することや、除去することも可能です。

Data Flow Diagramのメタモデルを開くには、CTRLキーを押しながらData Flow Diagramのテキストシンボルをダブルクリックします。Figure 1-1と同じメタモデルがDiagram Editorに開くでしょう。この動作を繰り返すことで、他の言語のメタモデルを確認できます。

2.3 新しいメタモデルを作り出す

次のステップは、新しいモデリング言語をグラフィカル・メタモデリング言語を用いて開発します。言い換えると、メタモデルを作り出すためにメタモデリング言語を使用します。全く白紙の状態から、ユースケースダイアグラムのグラフィカルなメタモデルを作ります。ユースケースダイアグラムを選んだ理由は、比較的知られていることと、コンセプトが少ないためです。MetaEdit+を用いれば、この程度であれば30分でメタモデルを作り出すことが出来ます。

2.3.1 新しいグラフタイプを作る

ユースケースのメタモデルのために、まずは新しいダイアグラムを作ります。メインウインドーの**Create Graph**ボタンをクリックします（あるいはGraphブラウザーで左マウスクリックで開くポップアップから）。 そうするとメタモデルのタイプの選択が現れます。今回は1つの言語を作るだけですので、Metamodel [GOPRR] を選んで **OK**をします。

次にグラフの名前を入力し（‘Use Case Diagram’）、Propertiesの領域でポップアップメニューから**Add Element…**を選びダイアグラムに与えたいプロパティ（‘Model name’ と ‘Documentation’など）を加えます。ここで開くダイアログから各プロパティの値を入力できます（以下、Figure 2-2参考）。必須の名前（Property name）、ローカル名（local nameモデリングツールで用いる名前）や、Datatypeには各プロパティのデータタイプも指定できます。これらのデータタイプについて詳しくは、MetaEdit+ Workbench User’s Guideにあります。

ここでは、2つのプロパティ‘Model name’のDatatypeにStringと、‘Documentation’のDatatypeにTextを設定します。ダイアログではプロパティごとに、初期値（default values）、一つだけ存在できるという唯一性のコンストレインツ（Uniqueness）も設定できます。‘Model name’には、Uniquenessに‘globally’を選びます（同じ名前のユースケースダイアグラムが出来ないようにするため）。
最後に各メタモデルのエレメントに対する、Descriptionに入力します。このDescriptionに書かれた内容

は、作成される言語、およびMetaEdit+でモデルを書く時にHelpから参照できます。‘Model name’のプロパティ設定ダイアログは、以下Figure 2-2のようになるでしょう。**OK** をして、ダイアログを閉じます。

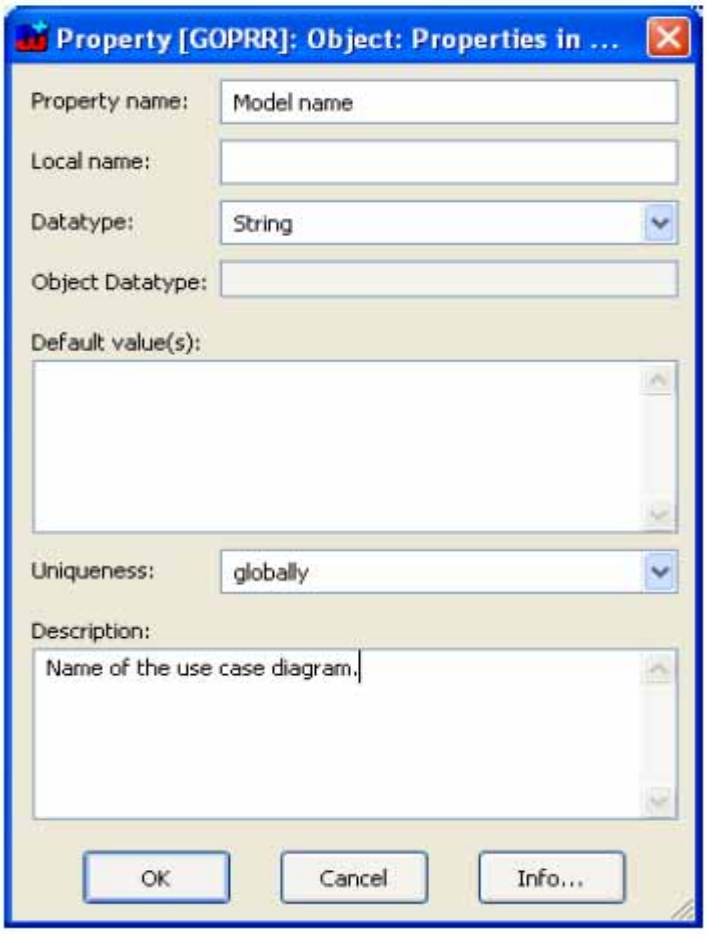


Figure 2-2. Dialog for adding a ‘Model name’ property for Use Case Diagram metamodel

‘Documentation’のプロパティも同様に入力します。そして、‘Use Case Diagram’の設定ダイアグラムを閉じると、空のダイアログエディタが開いてきます。

2.3.2 メタモデルに新しいオブジェクトを追加する

次は、モデリング言語で用いるオブジェクトを設定します。ユースケースダイアグラム内の、‘Use Case’, ’System’, ’Actor’です。’Actor’コンセプトから開始しましょう。ツールバーからObject [GOPRR]ボタン（水色四角、左から2番目）クリック、あるいはtypeメニューからこれを選択し、ダイアグラム上にクリックします。オブジェクトの詳細設定ダイアログが開くでしょう。

まず‘Actor’という名前をObject nameに与えます。そしてPropertysにて、各アクターが持ち得るプロパティを設定します。‘Actor name’というプロパティを前章のグラフタイプ同様の手順で設定できるでしょう。

Actorのコンセプトの設定で、既存のプロパティを再利用することも可能です。例えば、‘Documentation’という前章で作成したテキストのプロパティタイプは、Actorにも利用することが出来ます。Properties から **Add Existing…**を選択すると（**Add Element…**で作るのではなく）、開くダイアログから全ての使用可能なプロパティが閲覧できます。先程作った‘Documentation’をダブルクリックして、追加リストに表示させ、**OK**ボタンを押すことで、Propertiesにこれが追加されます。

‘Actor’オブジェクト設定には、モデルのコンセプトをDescriptionに記載することや、存在回数のコンストレインツを指定することもできます。この存在回数のコンストレインツの初期値‘N’は、複数のActorが1つのユースケースダイアグラムに存在できることを意味します。‘0’の場合、そのオブジェクトはアブストラクト（抽象）となり、Figure 1-1で紹介したData Flow Diagram例の“Abstract”オブジェクトと同様です。2つのプロパティを設定すると、ダイアログは、以下Figure 2-3のようになるでしょう。 **OK**を押すことで、作成されたオブジェクトがダイアグラムに追加されます。

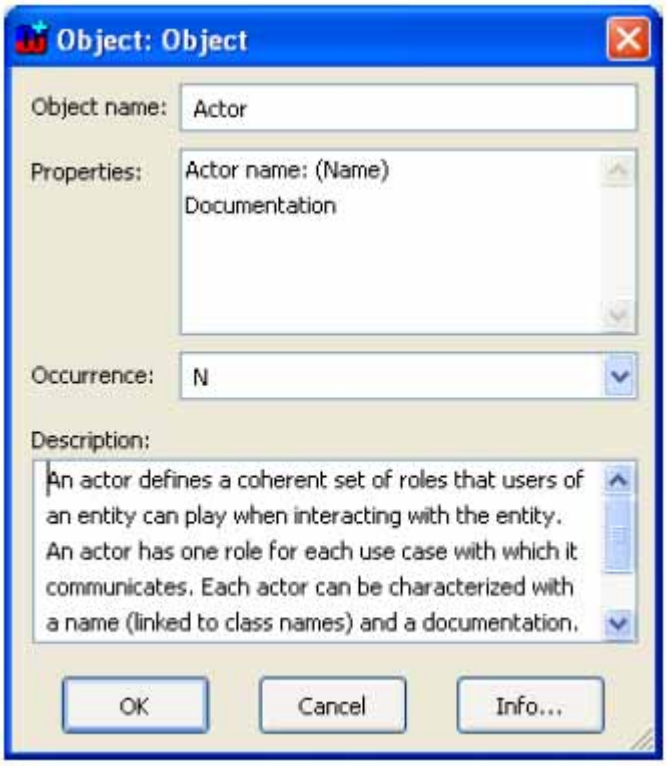


Figure 2-3. The specification of ‘Actor’ object

次に、ユースケースダイアグラムの他のモデリングオブジェクトである、‘System’ と ‘Use case’を続けて作成します。‘System’には、Occurrenceを1にして、各ユースケースダイアグラムが、1つのみSystemを表示できるようにします。複数表示にしたければ、この値を‘N’にします。
‘Use case’には、外部ファイルをリンク可能にするために、‘Documentation file’プロパティを作り、そのDatatypeに、External Elementを指定します。 以下のFigure 2-4.のようになるように、これらのオブジェクトを作成しましょう。

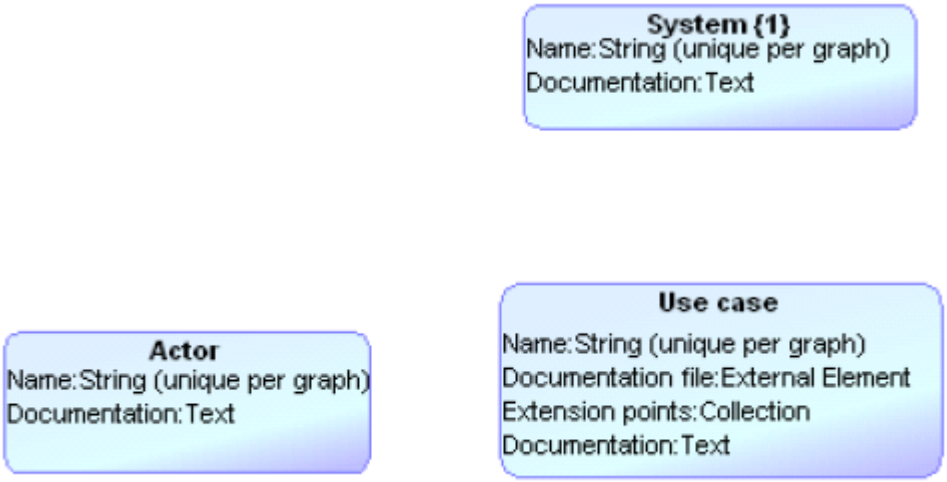


Figure 2-4. Modeling objects added to the metamodel of Use Case Diagram

2.3.3 オブジェクト間のバインディングを作成する

次に、オブジェクトタイプ間の接続を定義します。これは、オブジェクト間にBindingを作成することで実施します。Use caseオブジェクトとActorオブジェクト間の接続関係を、これらの間にBindingを定義しましょう。

Binding [GOPRR] (オレンジ色ひし形) ボタンをツールバーから選択し、2つのオブジェクトをクリックで結びます。(あるいは、1つのオブジェクトを選んでから、右マウスでのポップアップメニューから **Connect…**を選んで、別のオブジェクトに結ぶことも可能。また、他のコネクションを作成することもできますが、それに関して詳しくはMetaEdit+ User's GuideのDiagram Editor章を参考)

Binding関係を作るうえで、詳細設定のダイアログが開きます。Bindingは、MetaEdit+のGOPRRと同じコンセプトです。 オブジェクト間の接続にまつわる relationships とroleを定義します。ダイアログのBindingタブ内のRelationshipから、relationshipタイプや、その名前、プロパティが設定できます。オブジェクトの設定と同様です。ここでは、**Attach New Object…**を選択し、少なくても名前 (たとえば 'Association') を入力します。これには、Relationshipの名前を付ける必要があるでしょう。さらにオプションで追加のプロパティ、'Association name' なども設定できるでしょう。

Bindingには、2つのroleタイプ (FirstとLast) の設定も必要です。これはとなりのタブ (Figure 2-5) から行います。このユースケースダイアグラム例では、'Association role' を'Association' の時と同様に追加します。(ダイアログの'First role' タブをクリックし、Roleをダブルクリックして、Newで作成できるRoleタイプにRole nameとして'Association role' を設定) ここでRoleの名前は、必須項目 (mandatory property)。Roleタイプには濃度のコンストレインツ (cardinality) も設定が必要。この例では初期値1を使用 (同一関連内にuse case と actorが1つずつ存在するので)。 後で、別のBinding例で、1以外のcardinalityを紹介します。

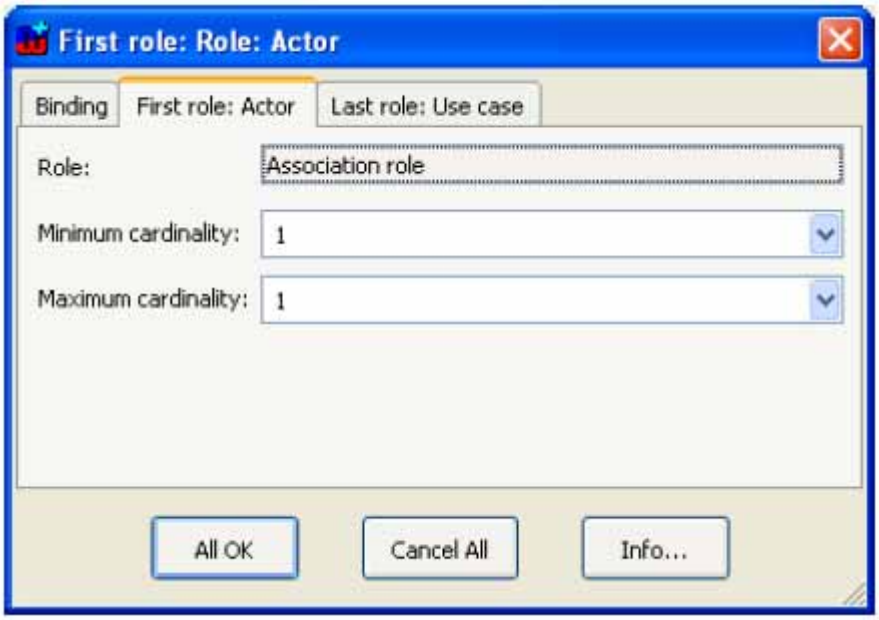


Figure 2-5. Creating binding between ‘Actor’ and ‘Use case’

ダイアログのLast Roleタブをクリックし、ここでのRoleには先程作成した‘Association role’を再利用します。Roleを右クリックしてポップアップメニューから、**Attach Existing Object…**を選択し、‘Association role’をダブルクリックで選びます。Cardinalitiesの値は、初期値1のままで。

最後に、Last Roleでは、‘Actor’ と‘Use case’の接続方向が選択できます。初期状態では一方向のみですが、**Can be drawn in both directions**を選ぶと、双方向でコネクションが生成できます。このオプションを選んで、**OK** でダイアログを閉じると、バインディングがダイアログ上に（‘Actor’ と‘Use case’オブジェクト間に）形成されます。

同様に、Figure 2-6にあるような、Generalization と Dependencyのリレーションシップも定義します。図を見ると、‘Generalization’ には、‘Superclass’ と ‘Subclass’ のroleタイプがあります。そして、‘Subclass‘ role のcardinalityは ‘1,N’となっていて、複数のサブクラスが、同じGeneralization リレーションシップ上に設定できるようになっています。

ユースケースダイアグラムでは、Dependencyのリレーションシップはuse case間のコネクションの拡張、使用を設定できます。これをメタモデルに定義するには、‘Dependency’リレーションシップに‘Stereotype’プロパティを追加します。このプロパティタイプは、‘use’ と ‘extends’という所定のリストを持っています。‘Dependency’に‘Stereotype’プロパティを加え、overridable listをデータタイプとして選択すれば、use caseをモデル化するときに、所定のリストから選択することや、独自の値を入力することができます。

ユースケースダイアグラムのBindingを完成させるために、‘Note’オブジェクトを追加し、Association bindingに加えます。これにより、追加のノートエレメントに関連を取らせることができます。存在するBindingに‘Note’オブジェクトを作成して追加するには、‘Association’リレーションシップを選択します。右マウスで表示されるポップアップから**Add a New Role…**を選択し、‘Note’オブジェクトに接続します。

これには、‘Note part’というRole名を設定し、minimum cardinalityは0にします。これにより、関連オプションとして、追加のノートができました。
同様のメタモデリングを行い、ユースケースモデリングに必要とあらば、他のBindingにもオプショナルなRoleを追加します。メタモデルは、以下Figure 2-6のようになるでしょう。

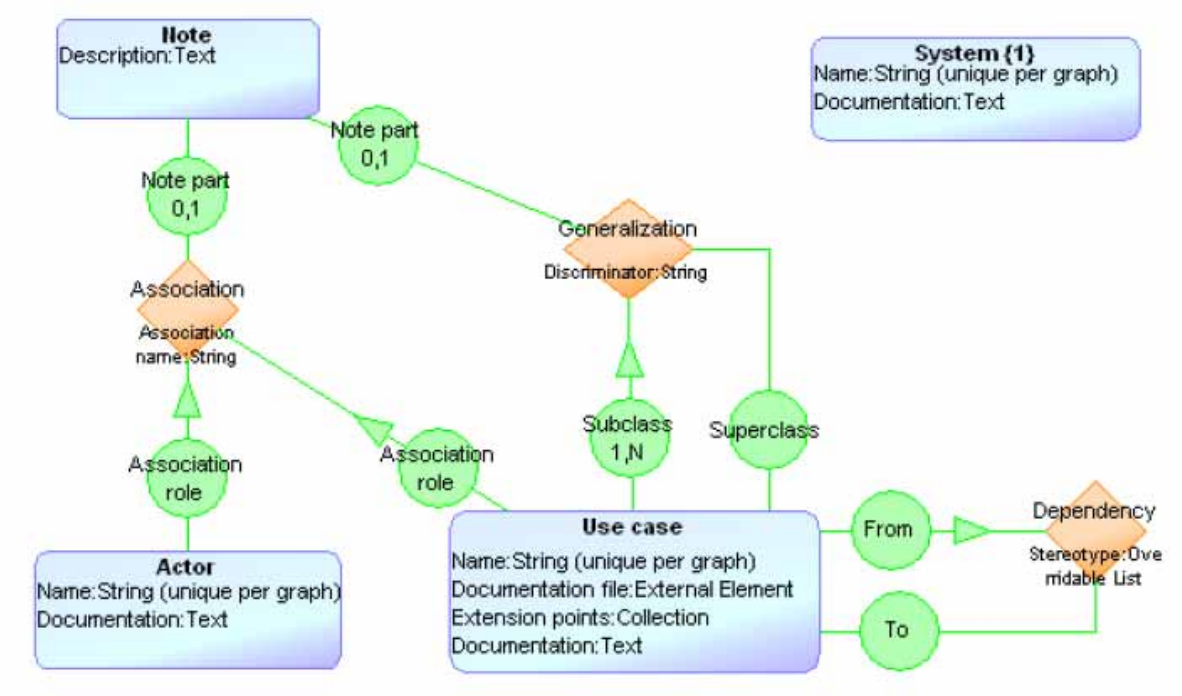


Figure 2-6. The graphical metamodel for Use Case Diagram

2.3.4 プロパティとしてオブジェクトを追加

メタモデルを完全なものにするために、Use Caseが取り得るAttributes と Operationsを定義する必要があります。特に、Use Case を Classとして設定したい場合に必要です。この目的で、‘Attribute’ と ‘Operation’という2つの新しいオブジェクトを作成し、これらを‘Use case’オブジェクトにproperty relationshipを用いて接続します。この接続は、Diagram Editor のツールバーの‘Prop’ボタンから、選択することができます。

Propertyの接続のために、任意のローカル名と、コンストレインツを設定できます。Use caseは複数のAttributes と Operationsを持つので、これらRelationship設定時のダイアログのNonProperty ofのタブ内にあるCollectionsを選択します。

‘Attribute’ と ‘Operation’オブジェクトには、追加のプロパティも設定できます。
ここで、これらオブジェクトの設定ダイアログでOccurrence の値を ‘0’することで、これらは抽象(abstract)となり、メインのモデリングコンセプトとしては使用されずに、‘Use Case’のみで使用可能となります。更に追加のオブジェクトをプロパティとして追加すれば、より複雑なモデリングコンセプトも作成可能です(Operationsのパラメータなど)

Figure 2-7では、ユースケースダイアグラム用の最終メタモデルを図示しています。これと同様の各種例は、インストールのGOPRRプロジェクトにございます。

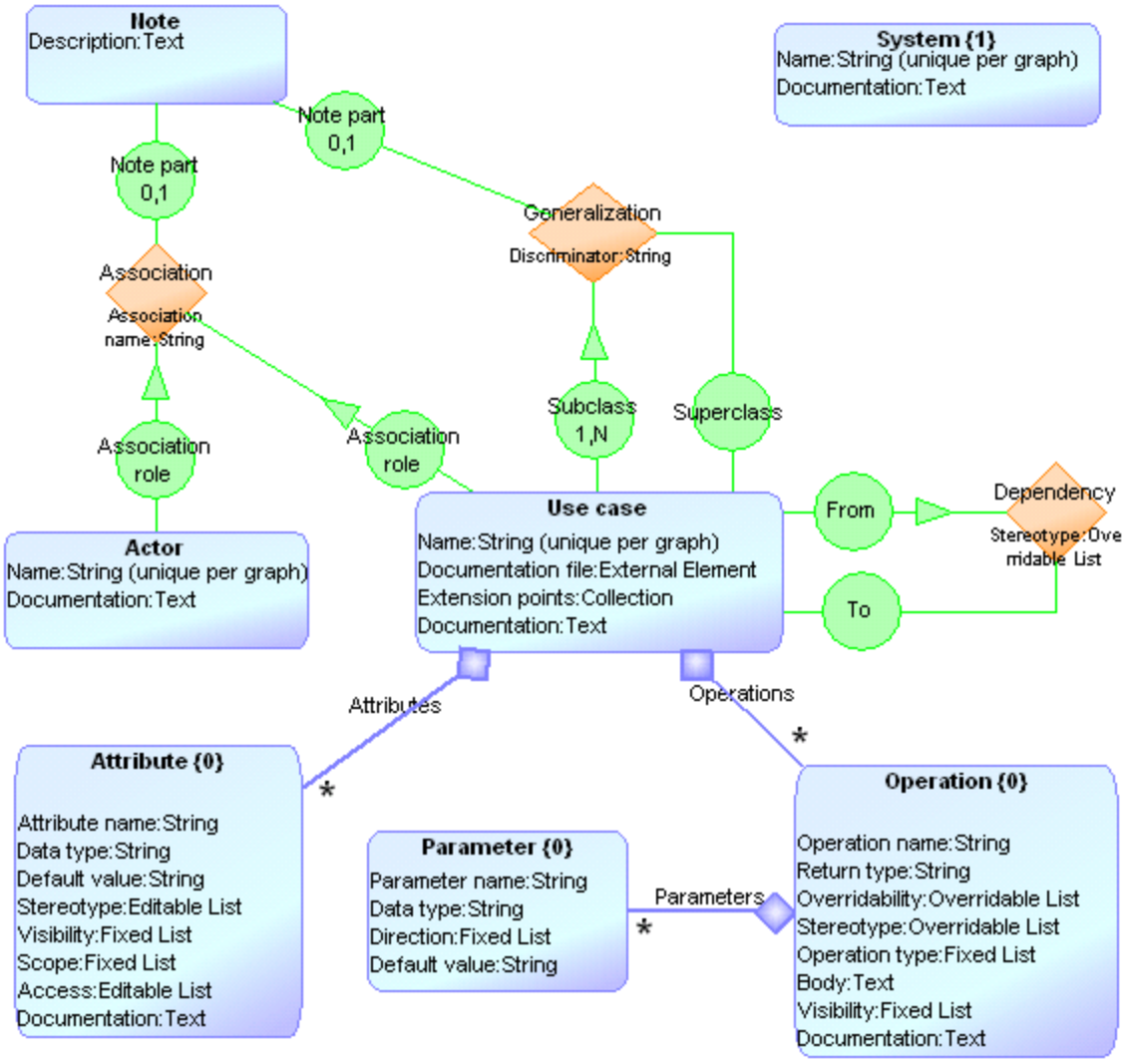


Figure 2-7. The complete metamodel for Use Case Diagram

2.3.5 言語の統合と、個々のグラフの統合

多くの場合、個々のグラフを1つに統合することや、複数の言語を統合することが必要になります。グラフィカル・メタモデリング言語では、GOPRRでサポートされる、そのようなexplosion と decompositionのリンクを作ることができます。Figure 2-1では、4つの言語間の、このような統合を例証しています。

ユースケースダイアグラム言語のデザインを完成させるために、各サブシステムオブジェクトが1つのサブグラフで表現できるように、言語の構造を規定しましょう。

はじめに、メインウインドーの**Create Graph**ボタンをクリックし、Metamodel for multiple graphs [GOPRR]を選んで、**OK**します。次にグラフの名前を入力し（例えば‘MyLanguage’）、**OK**します。
そして空のDiagram Editorが開きます（ユースケースダイアグラム言語をデザインした時と、少しだけ異なる）。 ここで統合するべき全ての言語の設定を、ツールバーのGraph [GOPRR]ボタンを選択してダイアグラム上でクリックすることで行えるようになります。これにより作図エリアにGraphシンボルが追加され、これは以下のように既存モデルをどのように統合するかを指南しています。

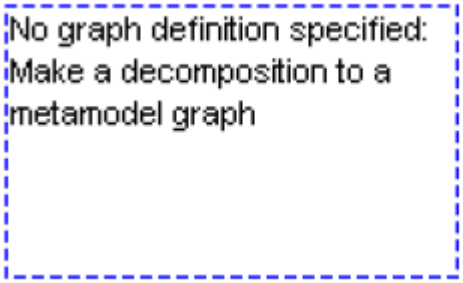


Figure 2-8. Graph added to the metamodel.

次に、このシンボルからのポップアップメニューからDecompositionsを選ぶことで、リンク可能なメタモデルリストが表示され、この例ではユースケースダイアグラムを選択し、リンクさせます。 （新しくメタモデルを作成すると、decompositionのリンク可能なメタモデルリストに追加されます） ポップアップメニューには、decompositionリンクの修正、削除、別メタモデルとの入替えなどの機能も用意されています。

Systemからユースケースダイアグラムにdecompositionをを設定するには、Systemのコンセプトをダイアグラム上に追加し、点線のグラフシンボル内に置く必要があります（Figure 2-9）。ここで、Systemのコンセプトとはユースケースダイアグラムの一部であることを意味します。
これを行うには、ダイアグラム上にSystemコンセプトを作るか、既存のSystemオブジェクトを追加します。そしてこれをGraphシンボルに、コピー・ペーストします。 このやり方の利点は、もしSystemの名前をどこかで変えると、同じオブジェクトを用いている全ての場所に、即座に反映されることです。

別の便利な（でも少し複雑な）方法は、トップグラフのTypesツールバーからObjectボタンを選び、好みの場所でshift-clickすると、既存のオブジェクトが選択できるダイアログが開きます（作図エリア内で、何も選択せずにポップアップを開いて**Add Existing...**を選ぶことと同じ結果）。ダイアログのリストには、グラフとそれらで用いられているオブジェクトが閲覧できて、あらゆる場所から再利用できるようになっています。
そして選択した結果は記録され、次回このダイアログを開いた時に右上のSelection Historyを選べば、それを再利用できます。

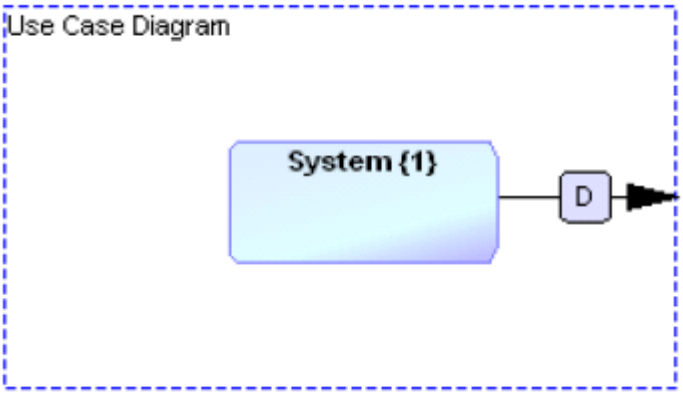


Figure 2-9. System is decomposed into another use case diagram.

言語構造を完成させるために、Systemオブジェクトから、ユースケースダイアグラムへの decompositionリレーションシップの設定は、ツールバーからDecomposition [GOPRR]ボタンを選択し、Systemオブジェクトから、ユースケースダイアグラムへ接続配線します。Figure 2-9
この構造により、ユースケースダイアグラムの階層を作成し、メインテナンスできるようになりました。

3 言語の定義をMetaEdit+に

ここまでに言語を作成し、そのコンセプト、プロパティ、コネクション、ルールを設定してきました。次は、このメタモデルをMXT形式のファイルに変換し、MetaEdit+ Workbenchにインポートします。

3.1 MXTの生成

MetaEdit+では、XMLベースでメタモデルのインポート/エクスポートをサポートします。このXML形式のメタモデルをMXT (MetaEdit+ XML Types file)と呼んでいます。このメタモデリング例では、グラフィカルメタモデルからXMLファイルを生成させることを紹介します。

ユースケースダイアグラムのグラフィカルメタモデルから、MXTファイルを生成する為に、ジェネレータを実行します。ダイアグラムエディタの**Graph | Generate…**から、表示される関連リストを選択します
(あるいは、ツールバー上のGeneratorボタンを押すことで)

メタモデルからMXTを生成させる為には、3つの選択肢があります。

・・‘Export graph type to MXT file’ (ツールバー上の‘MXT’ ボタンでも同じ)
　出力ディレクトリにMXTを生成。初期値では、MetaEdit+ のサブディレクトリ‘reports’に。
・・‘Export and Open MXT’
　MXTを生成し、ディフォルトのブラウザーで開く
・・‘Export and Build MXT’ (ツールバー上の‘Build’ ボタンでも同じ)
　MXTを生成し、MetaEdit+にメタモデルとしてインポートする

これらを行える権限の設定もあります。詳しくは、MetaEdit+ User's Guide の importing filesを参考。

複数言語を統合する為のメタモデリング言語にも、同様のジェネレータがあります。複数の言語は、同じMXTファイル内に包含されます。

この資料で作成したユースケースダイアグラムからMXTファイルを生成しましょう。上記3つの方法のうち、Buildボタンから実行すれば、MXTファイルは生成されてからMetaEdit+にインポートされます。それ以外の方法では、マニュアルで（MetaEdit+のメインウインドーからImportボタンを選び、インポートさせたいMXTファイルを選ぶ）、MetaEdit+にインポートさせる必要があります。

3.2 インポートされたメタモデルを進化させる

メタモデルをインポートすれば、MetaEdit+ Workbenchのメタモデリング機能を利用することができます。メタモデルを更に進化させ、Symbol Editorでノーテーションを与え、Generator Editorでモデルからドキュメント/コード/モデルチェックレポートなどを生成させたり、ダイアログやツールバーをカスタマイズすることなど。

メタモデルを修正・変更する場合、MetaEdit+ Workbenchのメタモデリング機能から、あるいはグラフィカルメタモデリングに戻って行えます。後者の場合には、修正・変更後のグラフィカルメタモデルから、MXTを再生成させる必要が有ります。

3.3 MXTジェネレータは拡張可能

MXTを生成させるジェネレータ機能は、MetaEdit+の他のジェネレータと同等のものです。これらは修正・変更して、メタモデルを他のフォーマットに変換することも可能です。**Graph**メニューの**Edit Generators**を選択して行います。1グラフから、あるいは複数グラフからMXTを生成させる2つのジェネレータがあります。

4 まとめ

この例では、グラフィカルメタモデリングについてご紹介しました。メタモデリング言語により、ドメイン・スペシフィック言語の基本構造をデザインし、他の言語まで統合することができます。

作成されたグラフィカルメタモデルは、モデリング言語としてMetaEdit+にインポートされます。そして、MetaEdit+は、様々なエディタ機能、ブラウザー、マルチユーザ対応などでモデリングをサポートします。メタモデルのインポートは、MXTフォーマット(MetaEdit+ XML Types)で行います。インポートされると、その言語（メタモデル）は、MetaEdit+ Workbenchを用いて、ノーテーション（表記シンボル）、ジェネレータ、追加のコンストレインツ、ルールなど拡張され、またモデリングをサポートする為に、ダイアログやツールバーはカスタマイズすることもできます。

メタモデリング言語は、MetaEdit+のGOPRRメタモデルから作られています。しかしながら、これはオープンなモデリング言語であり（MetaEdit+内の他のモデリング言語同様に）、グラフィカルメタモデリングのために、追加・変更可能です。メタモデリング言語や、ジェネレータを必要に応じて拡張されることは、全く自由です。

富士設備工業株式会社 電子機器事業部
〒591-8025 大阪府堺市北区長曽根町1928-1
Tel：072-252-2128 www.fuji-setsu.co.jp