MetaEdit+

Moving From Coding To Model-Driven Development:

Hands-On with MetaEdit+

The hands-on session at Code Generation 2009

Cambridge, UK. 16th-18th June 2009

17th June 2009
Juha-Pekka Tolvanen
Risto Pohjonen
Steven Kelly

MetaCase

MetaEdit+ ハンズオントレーニング 概要

モデルから完全なコード生成と、その生産性向上に興味を持っても、どのようにしてコーディング中心の開発からモデリングベースへ移行できるかは、良く知られていません。特に白紙状態から、抽象度を上げて高い生産性を得る（UML ではかなわない）独自のモデリング言語を作ることは、難しいと思われています。そこで、このセッションでは、MetaEdit+ の例題演習を介して、イテレーティブに、実用的な独自のモデリング言語が、コード資産を基にして、容易に実装できることを紹介します。

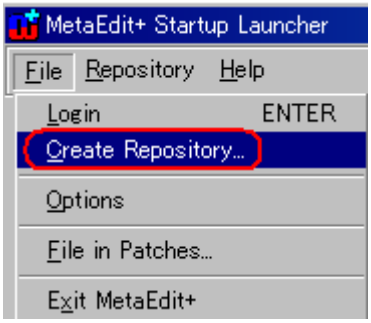
以下のステップで、始めはコードレベルの抽象度でモデリング言語（メタモデル）を作成し、それを試用し、イテレーティブに修正する（ルールを持たせる、抽象度を上げる）ことで、間違ったモデルを描くことの無い、より洗練されたモデリング言語へと進化させる仕組みを紹介します。(所要時間 3 時間)

注：この演習では、メタモデリング（モデル言語の作成）方法の理解を中心にしています。そのため、例題では簡単なコード生成に限定しています。さらなる次のステップとして、より複雑なコード生成（製品レベルの）の仕組みに興味を持たれるなら、MetaEdit+ インストール内の“デジタル腕時計”、“S60 携帯“ などのサンプルと資料を参考ください。(自動生成させる言語の仕様（AUTOSAR など)、あるいはフレームワークがあれば、容易であることは、この資料からでもイメージできるようにはなっていますが)

＜Version 0：Unityped Modelling Language＞
－コードコンセプト、コマンドレベルでモデリング
＜Version 1：ドメインスペシフィックな コンセプトを加える＞
－新しいメタモデル、表記、セマンチックス
－コードジェネレータ
＜Version 2：論理的にグループに分割する＞
＜Version 3：モデルの再利用＞
＜Version 4：よりハイレベルなドメインコンセプト＞
－新しいメタモデル
－コードジェネレータ
＜まとめ＞

下記は、ハンズオンで使用するファイルです。デスクトップなどに保存してください。
- GOPRR.mxt　：メタモデリング言語の定義ファイル
- *.svg　：事前に定義されたグラフィカルシンボルファイル（図示記号）
（Blow.svg、Command.svg、Clean.svg、Filter.svg、Move.svg、Suck.svg、To.svg）

[リポジトリの作成とログイン]
■リポジトリを作成してログインする
１．MetaEdit＋を起動し、メニュー“File->Create Repository”を選択する。



２．Creating new databaseダイアログに、下記を入力してOKをクリックする。

Database name: **CodeGen09**

Database root directory: **CodeGen09**

Your name: **user**

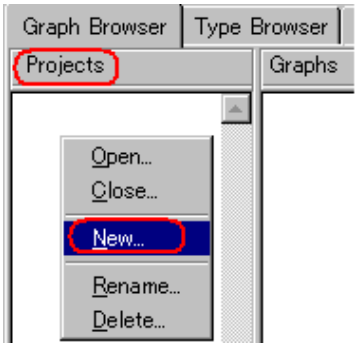
Your password: **user**

３．パスワード確認ダイアログでは、再度パスワードを入力する。

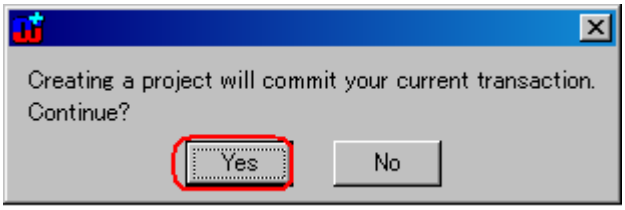
４．新たに“CodeGen09”を作成するか聞いてくるので、OKして作成する。

■新しいプロジェクトの作成

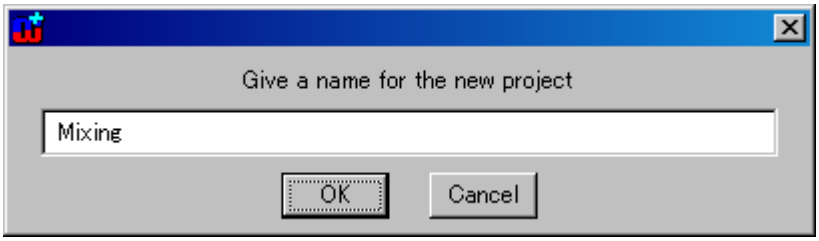
１．Projects内で右クリックし、“New”を選択する。



２．現在の処理をコミットするためのプロンプトが表示されるので、“Yes”をクリックする。



３．新しいプロジェクトの名前として、“Mixing”を入力して、OKをクリックする。

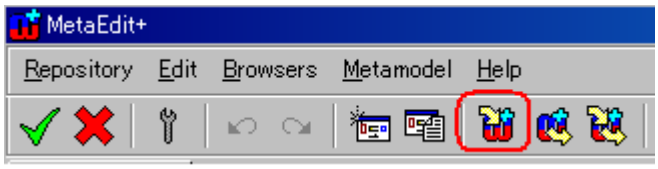


4．プロンプトが表示されたらOKをクリックする。

5．Commitボタンをクリックして保存する。

■グラフィカルメタモデリング言語のインポート

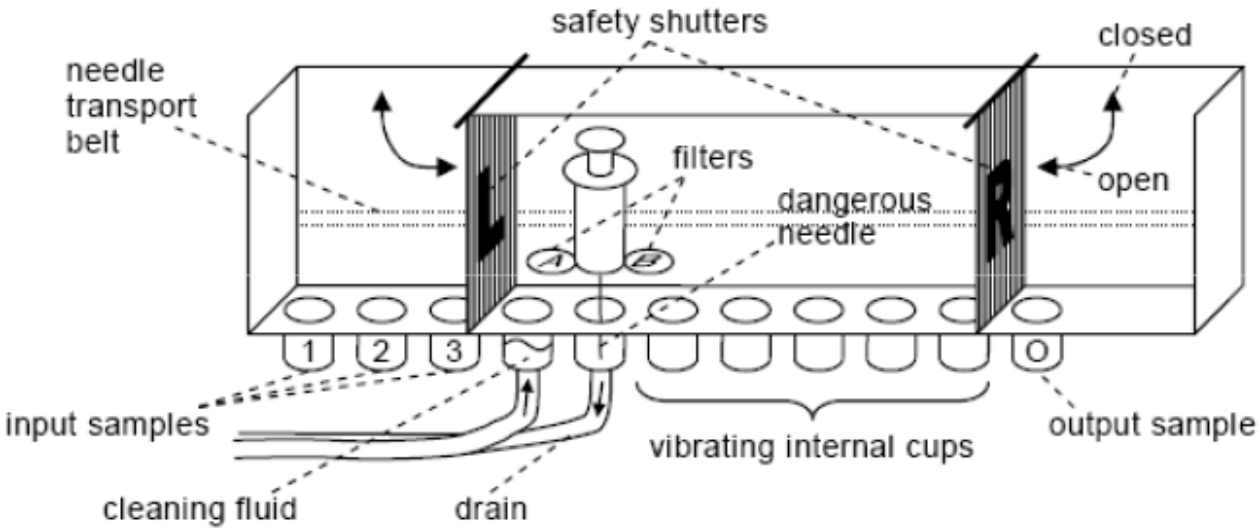
1．インポートボタンをクリックする。



2．ダイアログから、“GOPRR.mxt”を選択して開く。

3．Commit ボタンをクリックして保存する。

■［例題］：Medical Mixing Machine（医療用ミキサー）



“take from the second cup
5 units with filter A and
put 2 units to cup 6
and 3 units to cup 7 and
then clean the needle”

```
01 move(-3); filt(1); suck(5);
02 move(4); filt(0); blow(2);
03 move(1); blow(3);
04 move(-3); suck(30);
05 move(1); blow(30);
```

＜それぞれの動作と、それに対する命令（右）＞

（説明）“filter A”を使い、”input sample”のカップ 2 から 5 滴分吸い上げ、
カップ 6 に 2 滴、カップ 7 に 3 滴入れ、そして注射針を洗浄する”

＜Version 0：Unityped Modelling Language＞

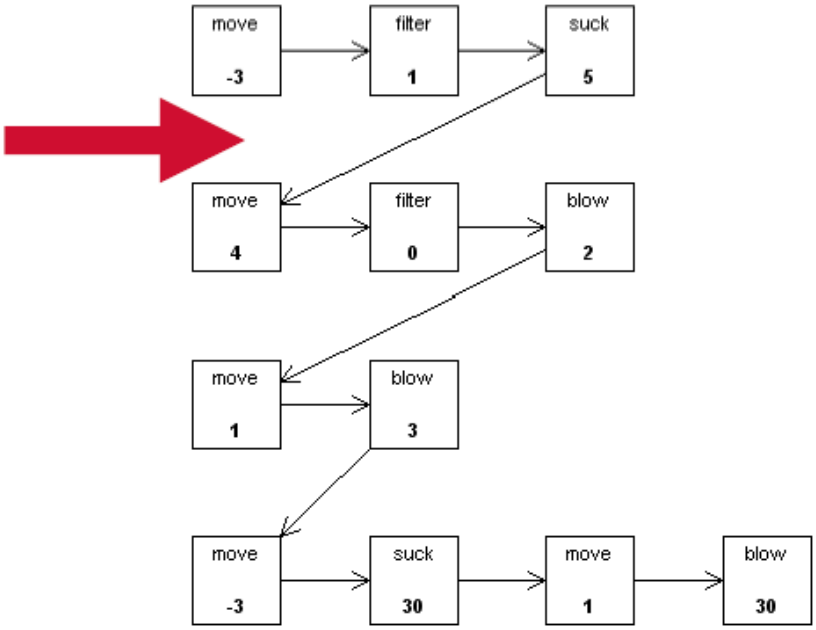
グラフィカルにメタモデル（モデル言語）を定義できる MetaEdit+ の機能（グラフィカル GOPRR）を用いて、例題の命令と実行フローからメタモデルを作成し、それを用いてアプリモデルを作成します。

■Version 0 のモデリング言語を以下のコンセプトで作成

－１つの命令ごとに、１つの box

－実行フローはリレーションシップとして矢印

```
move(-3);
filt(1);
suck(5);
move(4);
filt(0);
blow(2);
move(1);
blow(3);
move(-3);
suck(30);
move(1);
blow(30);
```



[GOPRR のコンセプト]

MetaEdit+ には、グラフィカルにメタモデルを定義するための、“グラフィカル GOPRR”という専用のモデリング機能があります。この GOPRR は、以下のモデリング部品の頭文字です。

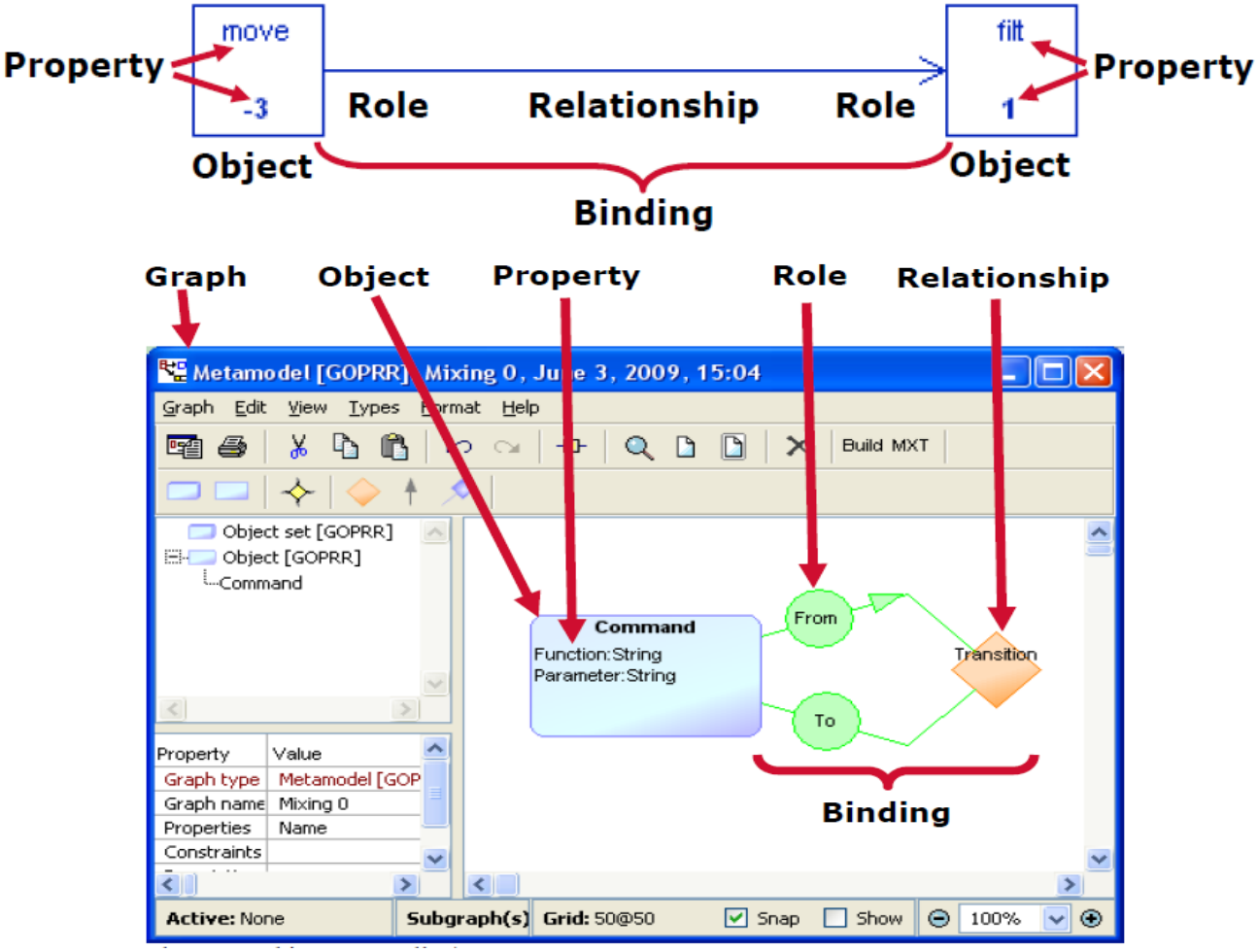
■ **Graph**　ひとつの独立したモデル
– ダイアグラム（図）として表記

■ **Objects**　Graphの主な要素
–箱または丸などを用いて表記

■ **Property**　各要素を特徴づける属性
–ラベルとして表記

■ **Relationship**　オブジェクト同士を接続
–接続間のラベルとして表記

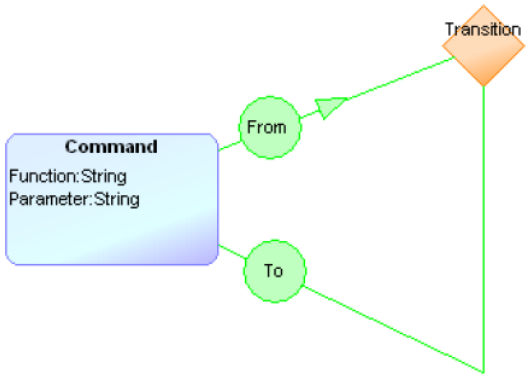
■ **Role**　relationship内でオブジェクトを接続
– 線や矢印として表記
– **Binding** は、**Relationship**, **Roles** や **Objects**　を接続



＜グラフィカル GOPRR のイメージ図＞

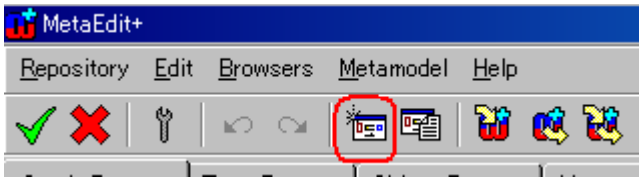
最初のメタモデリング（グラフィカルメタモデリング）

■ Command オブジェクトは、パラメータを用いたファンクションコール（move、filter、suckなど）
■ Transition リレーションシップや From、To ロールは、命令の処理手順などを定義

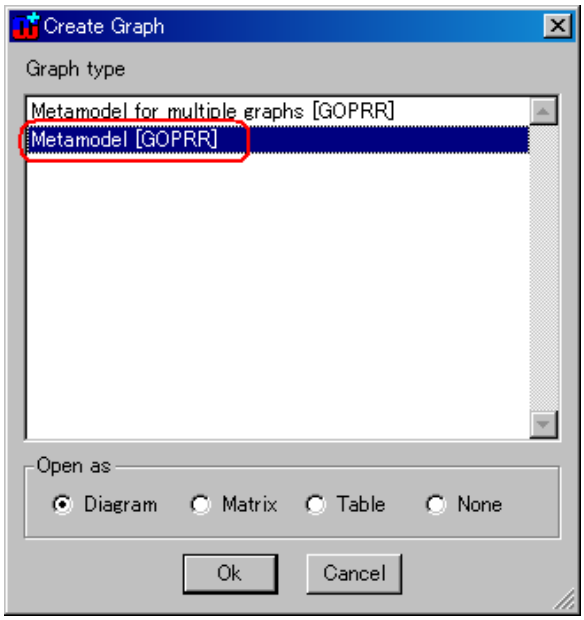


[Graph を作成]

１．ツールバーで、Create Graph ボタンをクリックする。

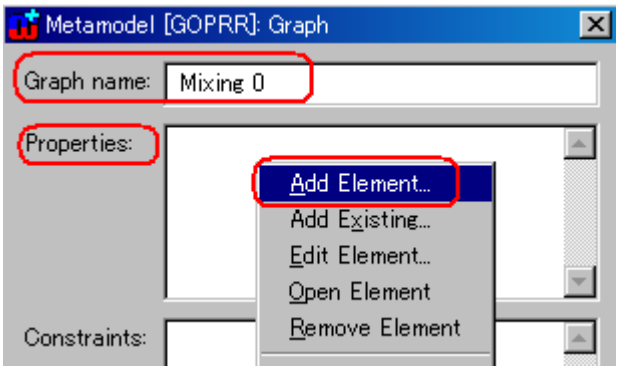


２．Create Graph ダイアログが開くので、リストから“Metamodel[GOPRR]”を選択し、OK する。

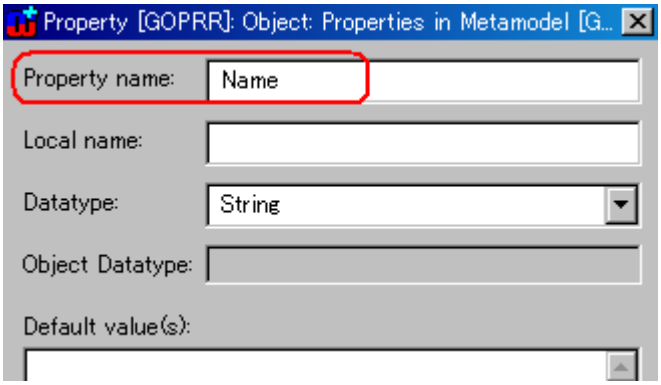


３．Graph を定義するダイアログが開くので、Graph name として、“Mixing 0”を入力する。

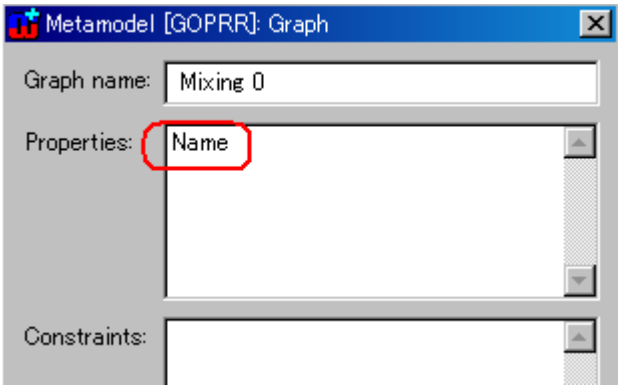
４．Properties 内で右クリックし、“Add Element”を選択すると Property を定義するダイアログが開く。



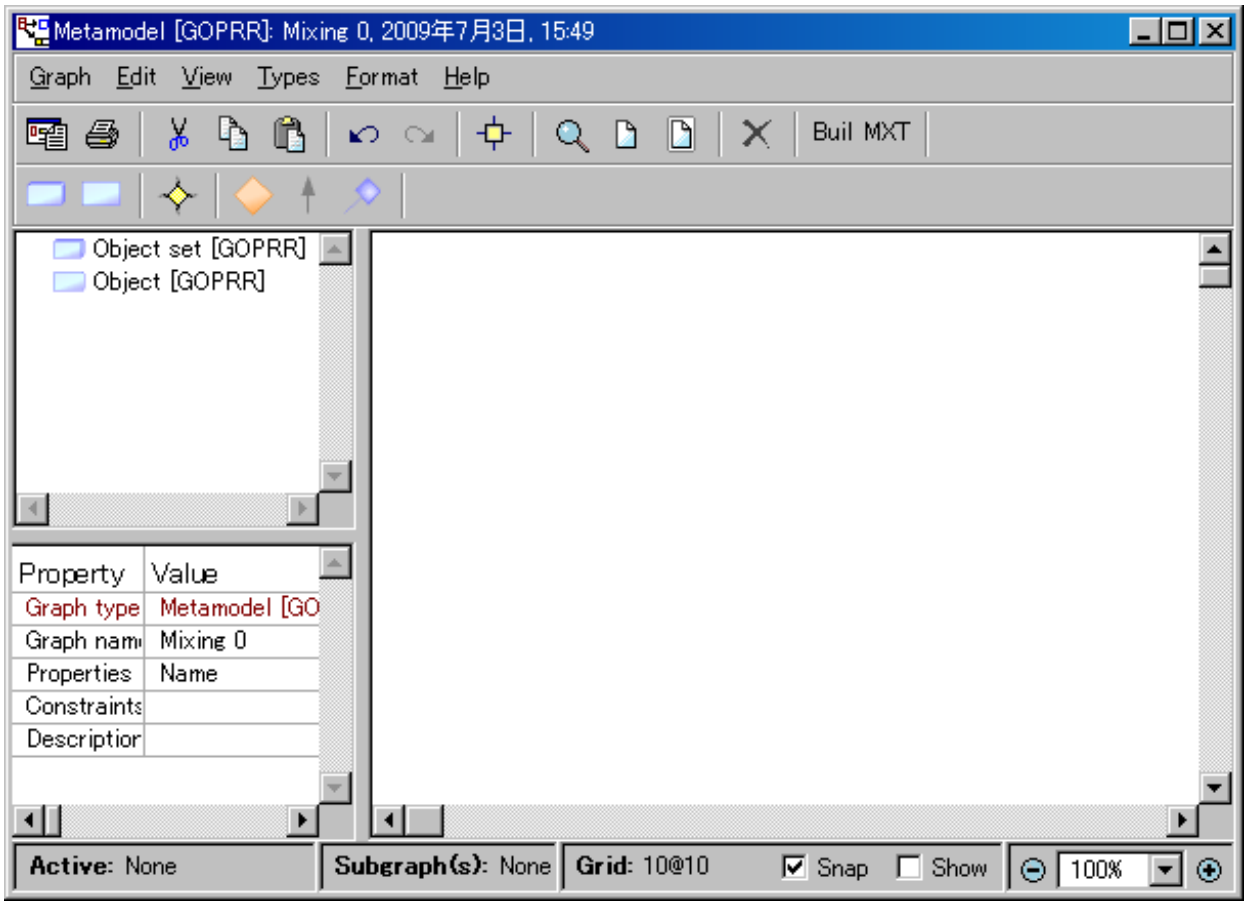
5．Property name として“Name”を入力し、OK をクリックする。



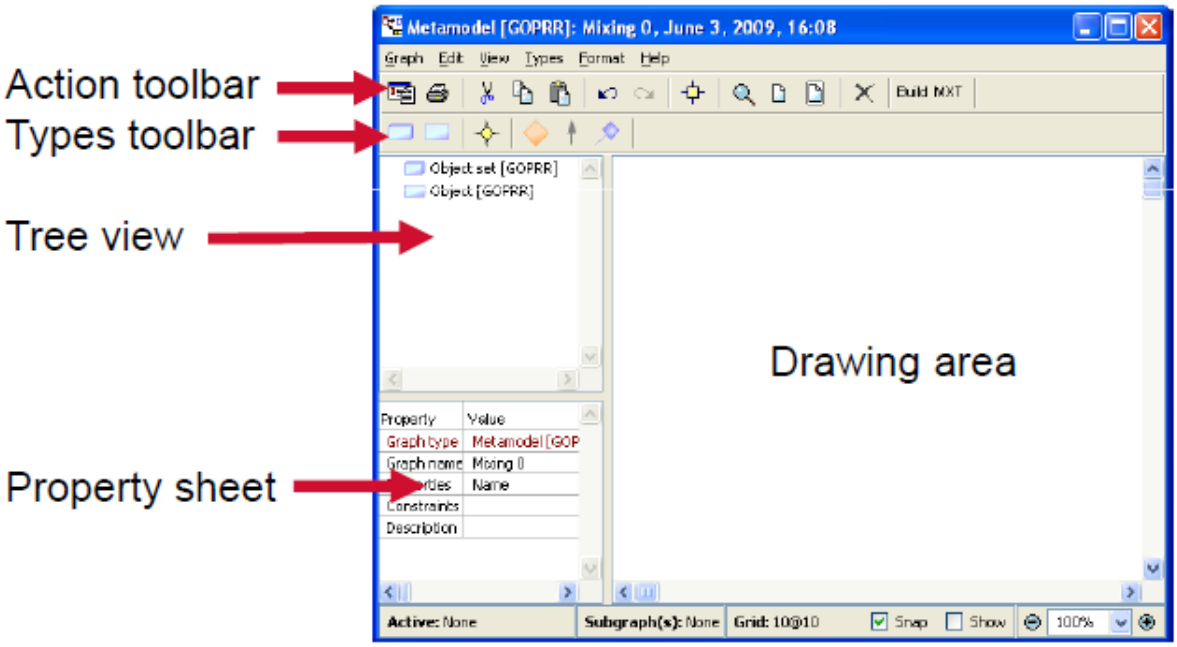
6．Graph を定義するダイアログは、以下のようになる。OK をクリックする。



7．新しい GOPRR メタモデルのためのダイアグラムエディタが開く。



■ ダイアグラムエディタで、グラフィカルメタモデルを定義。



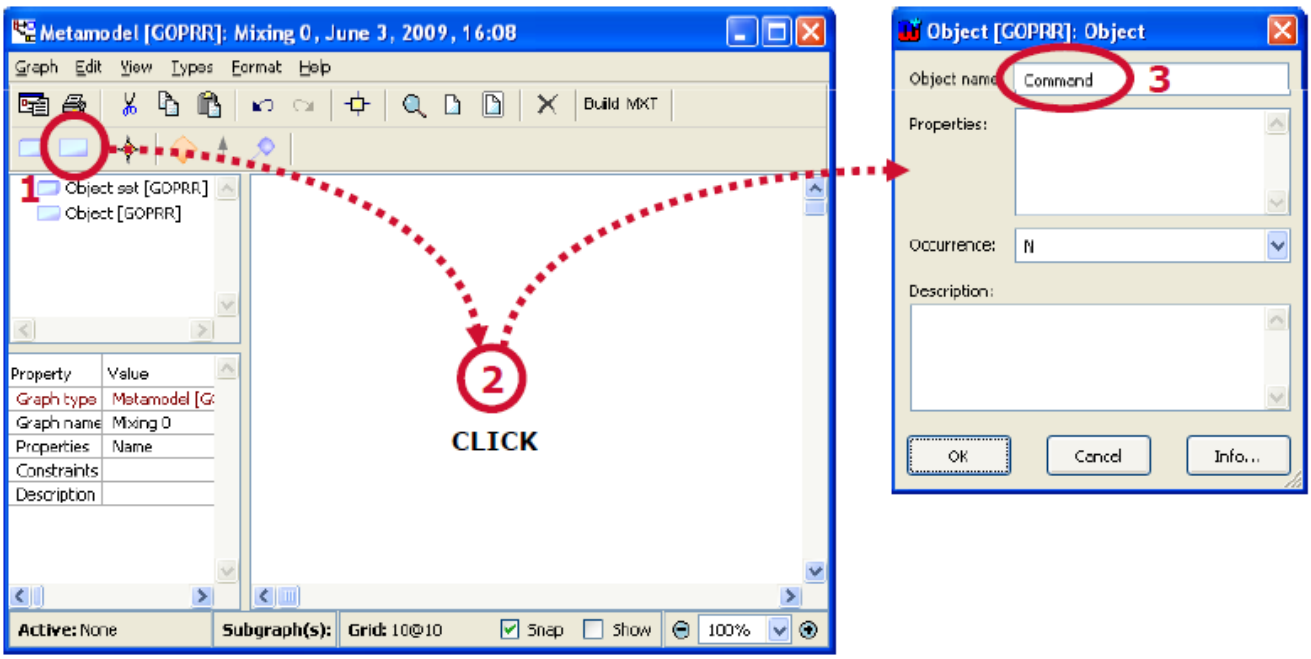
[グラフィカルメタモデルの定義]

・Object の作成

1．ツールバーで Object をクリックする。

2．描画エリアでクリックすると、Object 定義ダイアログが開く。

3．Object name として、“Command”を入力する。

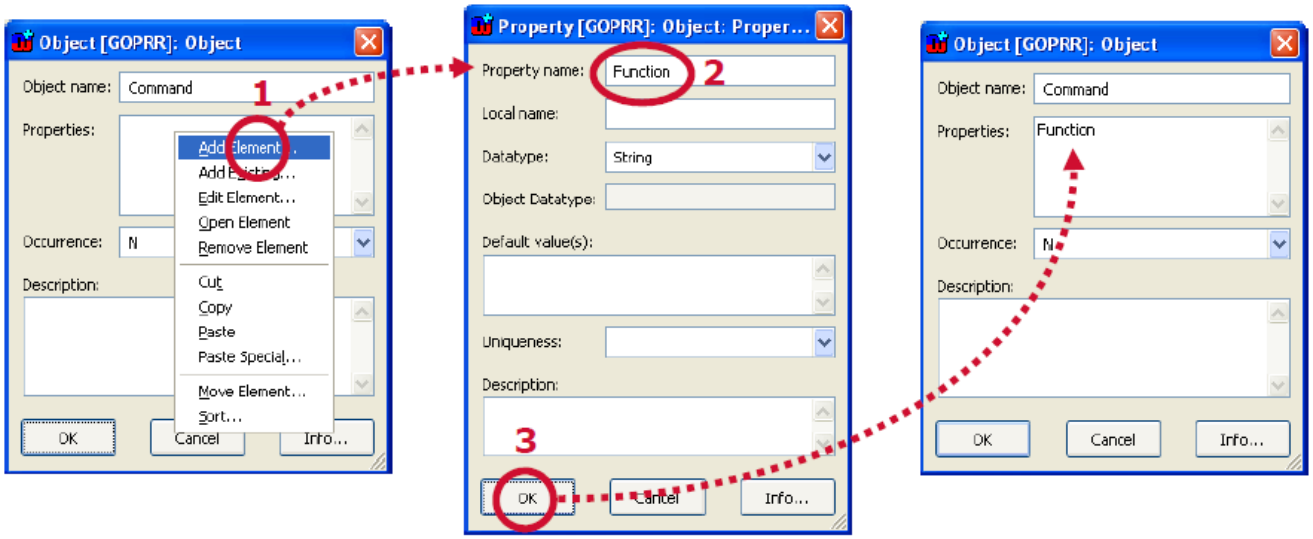


続けて、Property を定義。

1．上記 Object 定義ダイアログの Properties で右クリックし、“Add Element”を選択する。
Property 定義ダイアログが開く。

2．Property name として、“Function”を入力し、OK をクリックする。

3．Object 定義ダイアログの Properties リストに新しい Property として“Function”が追加される。

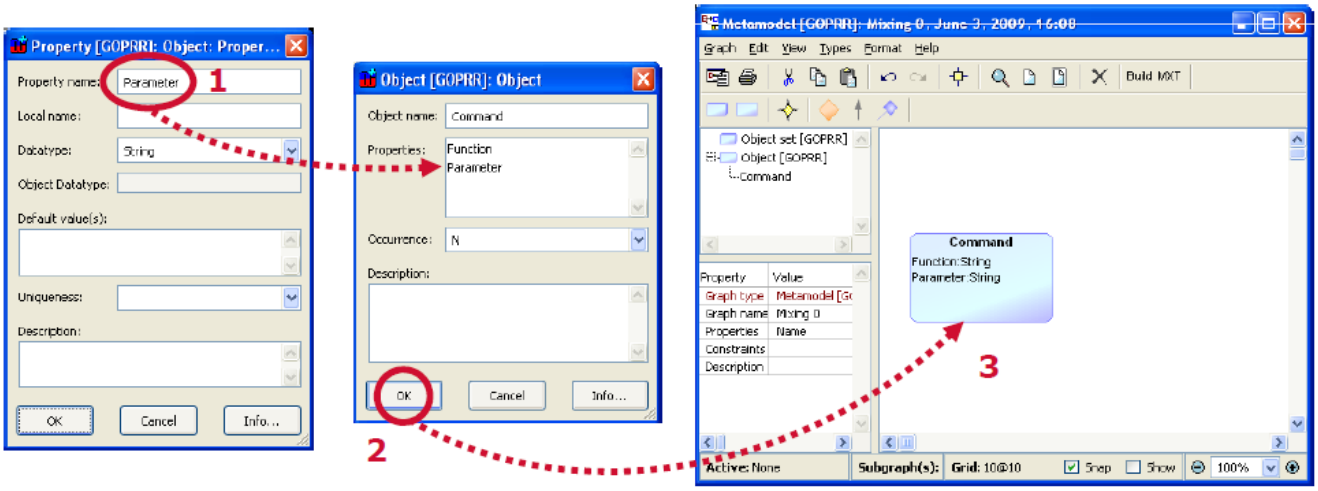


続けて、別の Property を定義。

1．Command に対して、上記 Function と同様に、“Parameter”という名前の Property を作成します。

2．上記 Function の手順１，２と同様です。

3．作成できたら、Object 定義ダイアログで OK をクリックする。ダイアグラムエディタに Command オブジェクトが現れます。

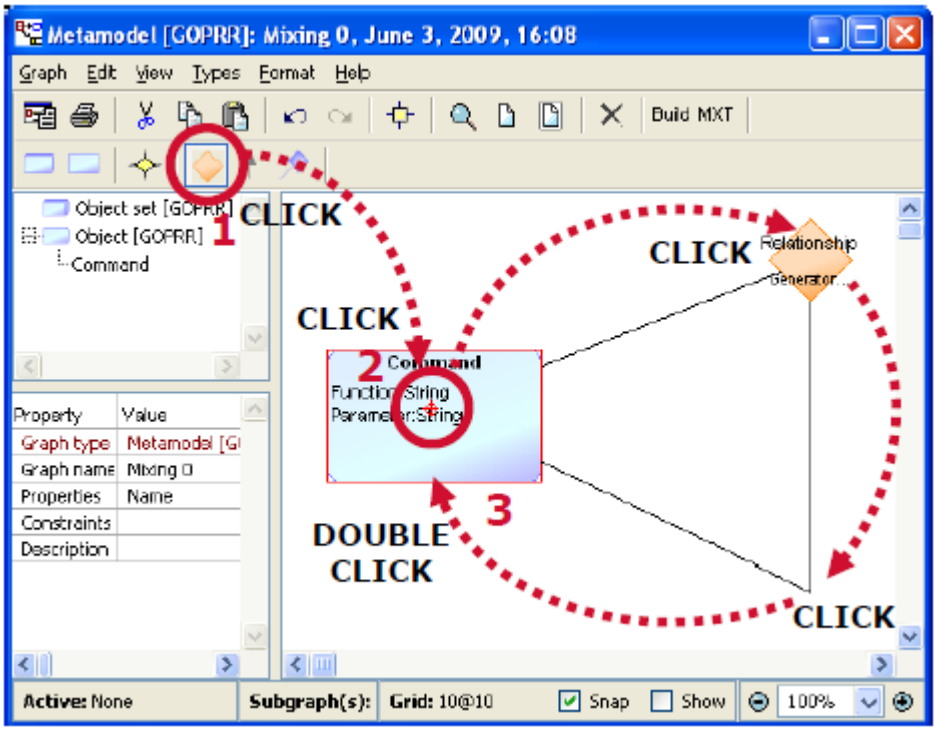


次に、Binding を作成。

1．ツールバーで Binding をクリックする。

2．Command オブジェクト上でクリックし、Command オブジェクトの外側で２つの折点を作成するようクリックする。（下図参照）

3．そして、Command オブジェクトでダブルクリックすると、Binding 定義ダイアログが開く。

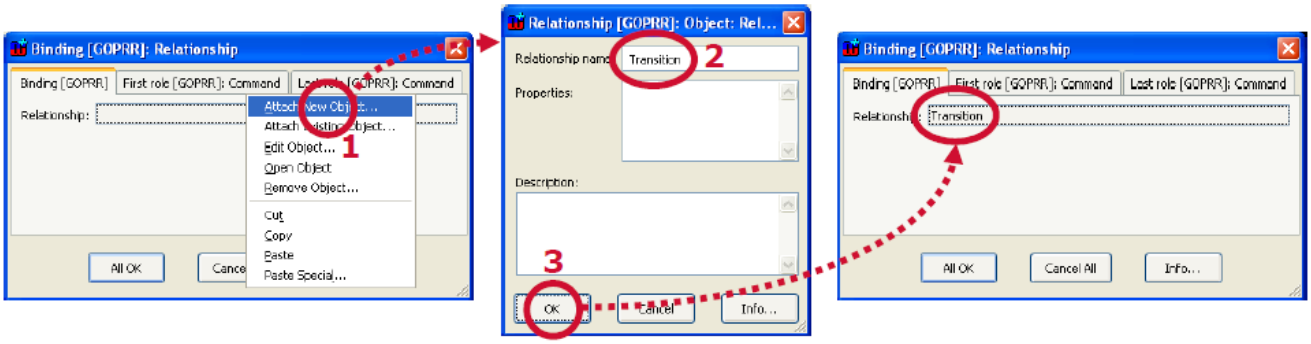


続けて、Relationship を作成。

1．Binding 定義ダイアログの Relationship フィールドで右クリックし、“Attach New Object”を選択する。Relationship 定義ダイアログが開く。

2．Relationship name として、“Transition”を入力し、OK をクリックする。

3．Binding 定義の一部として、“Transition”リレーションシップが表示される。
まだ、All OK ボタンを押してはいけません。



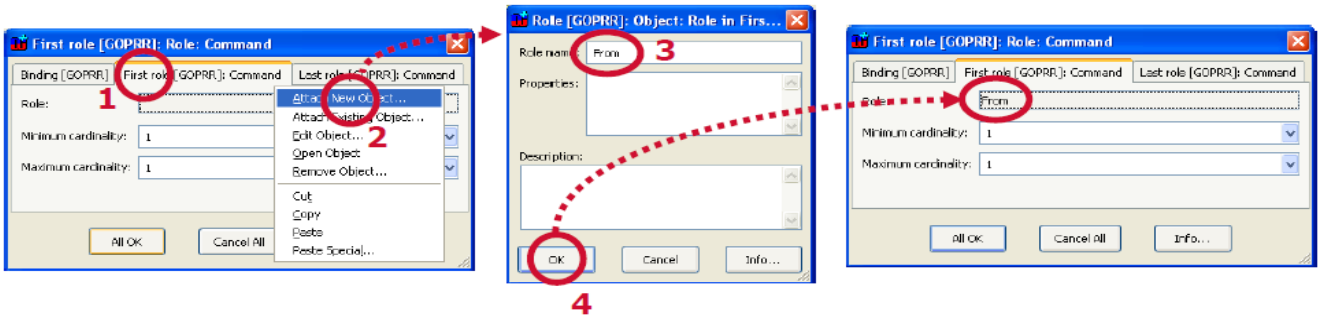
続けて、Role を作成。

1．Binding 定義ダイアログの First role タブを選択する。

2．Role フィールドで右クリックし、“Attach New Object”を選択する。Role 定義ダイアログが開く。

3．Role name として、“From”を入力し、OK をクリックする。

4．Binding 定義の一部として、“From”ロールが表示される。
まだ、All OK ボタンを押してはいけません。



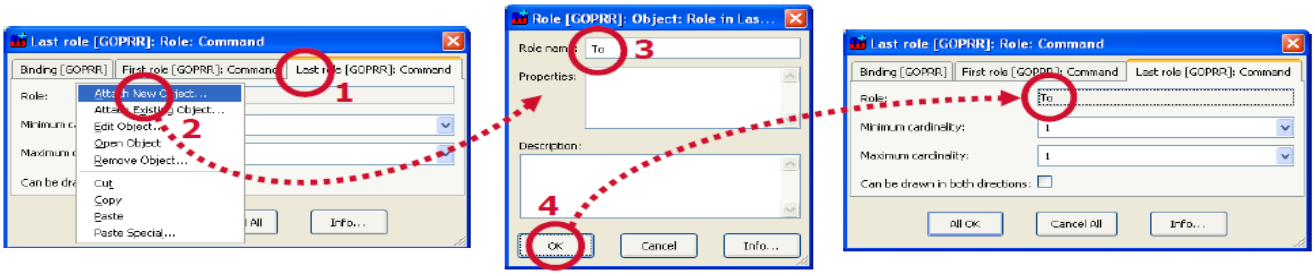
続けて、別の Role を作成。

1．Binding 定義ダイアログの Last role タブを選択する。

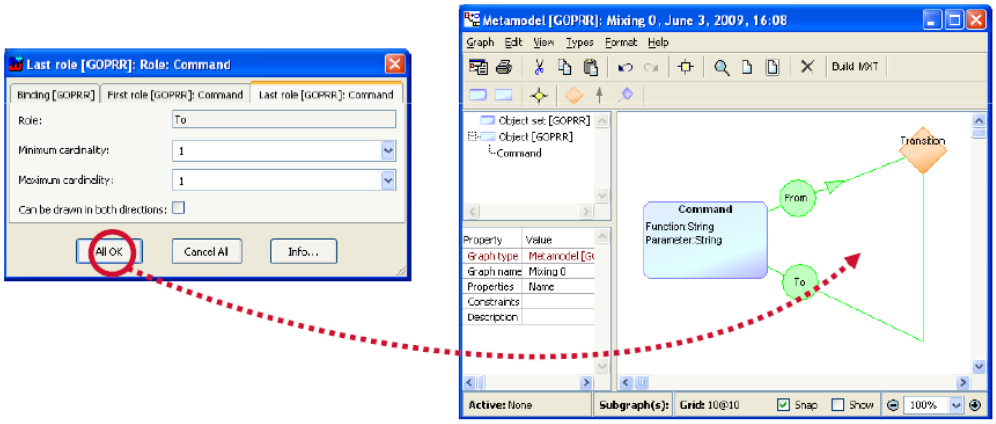
2．Role フィールドで右クリックし、“Attach New Object”を選択する。Role 定義ダイアログが開く。

3．Role name として、“To”を入力し、OK をクリックする。

4．Binding 定義の一部として、“To”ロールが表示される。

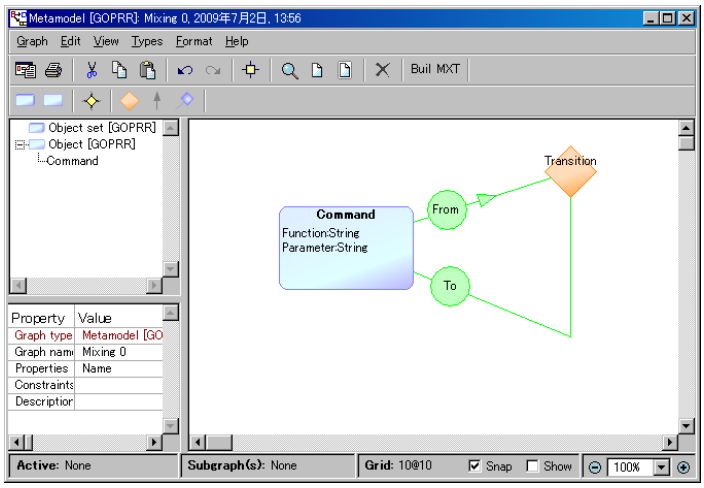


ここで、Binding ダイアログの All OK ボタンをクリックすると、ダイアグラムエディタに新しい Binding が表示されます。



最初のグラフィカルメタモデルが完成しました。

■ Commandオブジェクトは、パラメータを用いたファンクションコール（move、filter、suckなど）
■ Transition リレーションシップや From、To ロールは、命令の処理手順などを定義

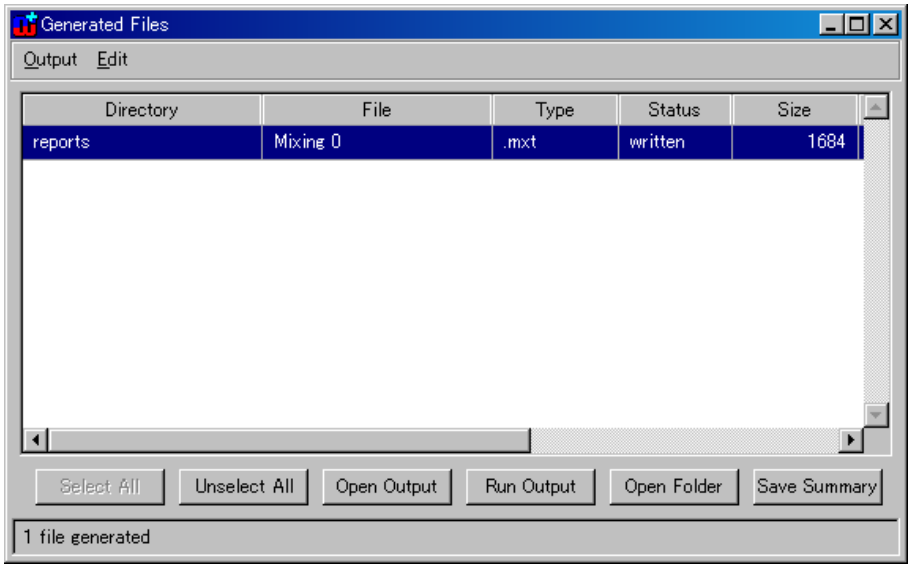


[グラフィカルなメタモデルをビルド]
上記で作成したグラフィカルメタモデルを、グラフィカル GOPRR の機能からビルドすることで、メタモデル（モデリング言語）環境が構築され、アプリケーションモデルが描けるようになります。

１．MetaEdit+メインウインドウで Commit ボタンを押し、メタモデルを保存する。

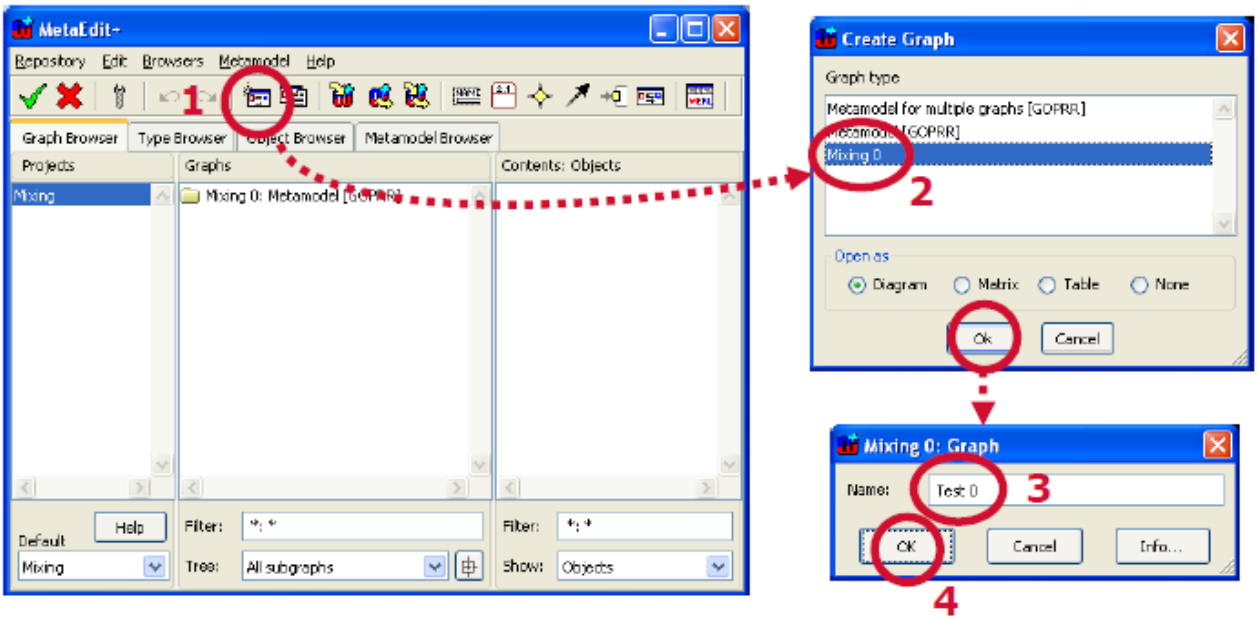
２．ダイアグラムエディタで、アクションツールバーの Build ボタンを押す。

３．MetaEdit+は、自動的にメタモデルをビルド
４．下記 Generated Files ダイアログが表示されるので、このダイアログを閉じる。

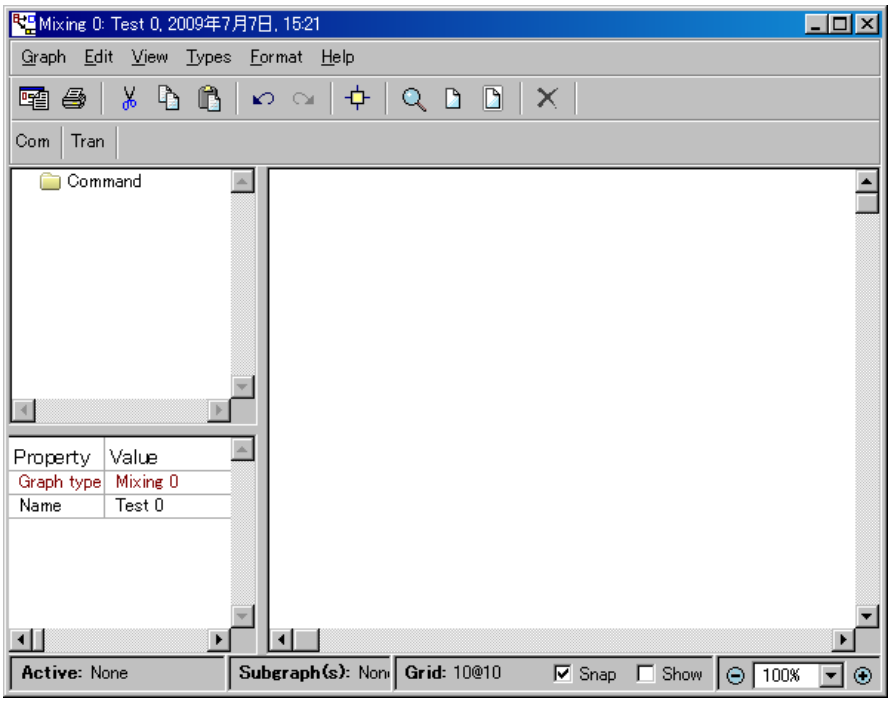


[新しいグラフ（アプリケーションモデル）の作成]

1．MetaEdit+メインウインドウで、ツールバーの Create Graph ボタンを押す。
2．Create Graph ダイアログから Graph タイプとして“Mixing 0”を選択し、OK をクリックする。
3．新しいグラフの名前を入力するダイアログが開くので、“Test 0”を入力する。
4．OK をクリックすると、ダイアグラムエディタが開く。



■ 基本的な動作は、グラフィカルGOPRRと同様
■ タイプツールバーには、今回作成した新しいモデル言語のオブジェクトタイプである、“Command”と“Transition”が入っている。



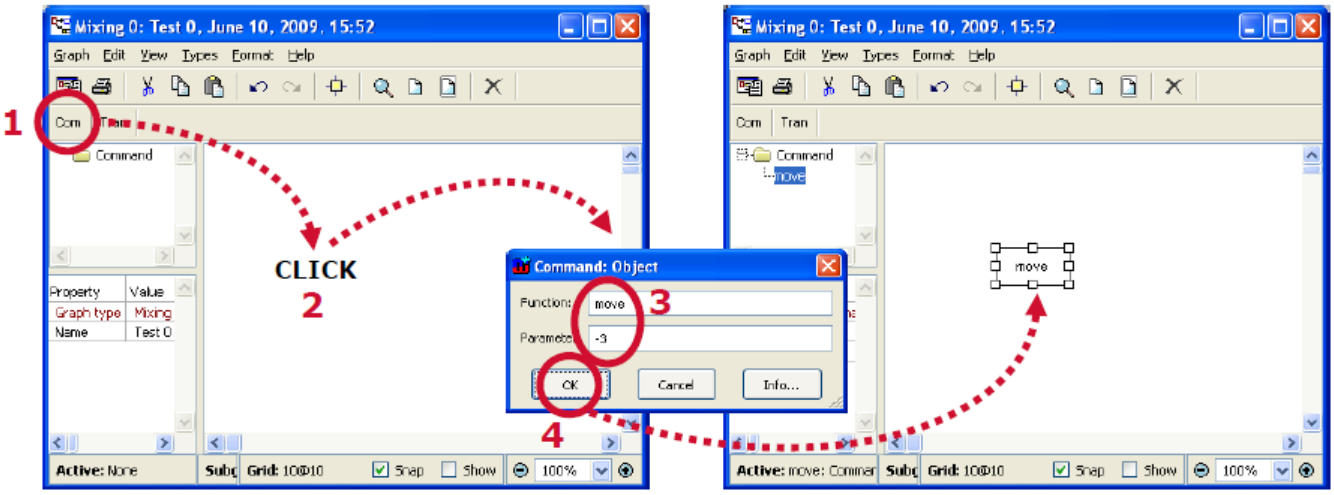
・Object の作成

1．Ctrl キーを押しながらタイプツールバーで、“Command”をクリックする（連続して Object を作成するため）。

2．Command オブジェクトを作成するため、描画エリアでクリックする。

3．プロパティダイアログが表示されるので、Function に“move”、Parameter に“－3”を入力する。

4．OK を押すと、新しい Object が、描画エリアに作成される。

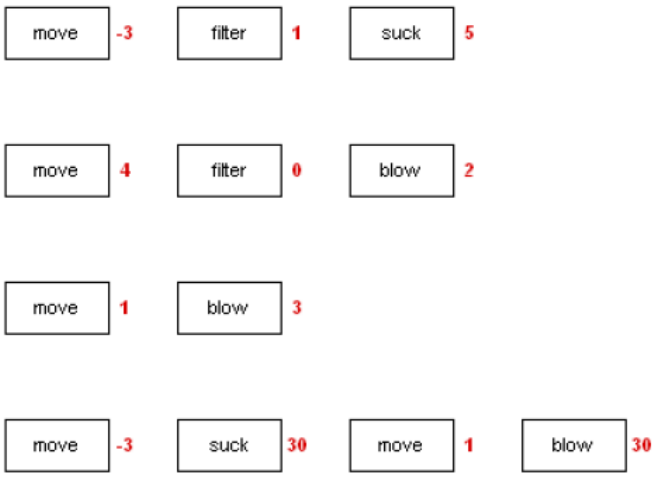


5．続けて、描画エリアをクリックするとプロパティダイアログが表示されるので、別の Object を作成する。

6．下図のように、Function の名前と Parameter の値を入力する。（赤字は、パラメータの値です）

7．上記 5 , 6 を繰り返し、全ての必要な Object を作成する。

8．全 12 個の Object を作成したら、描画エリアで右マウスボタンを押す。（Object 生成終了させる）

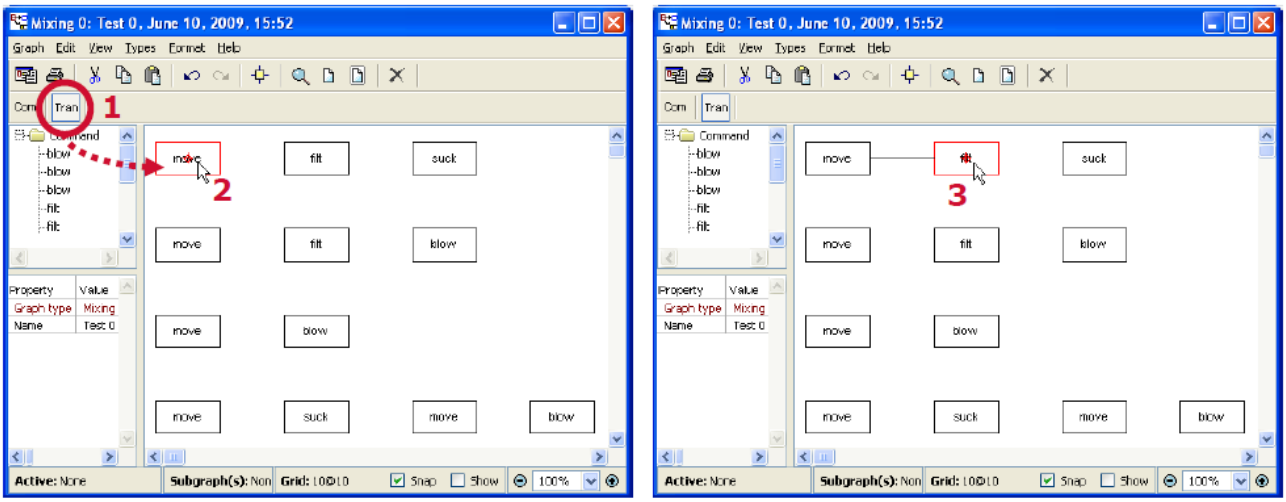


・Binding の作成

1．Ctrl キーを押しながら、タイプツールバーで、“Transition”をクリックする（連続して Transition を作成するため）。

2．最初の Object（move）をクリックし、押した状態のままにする。

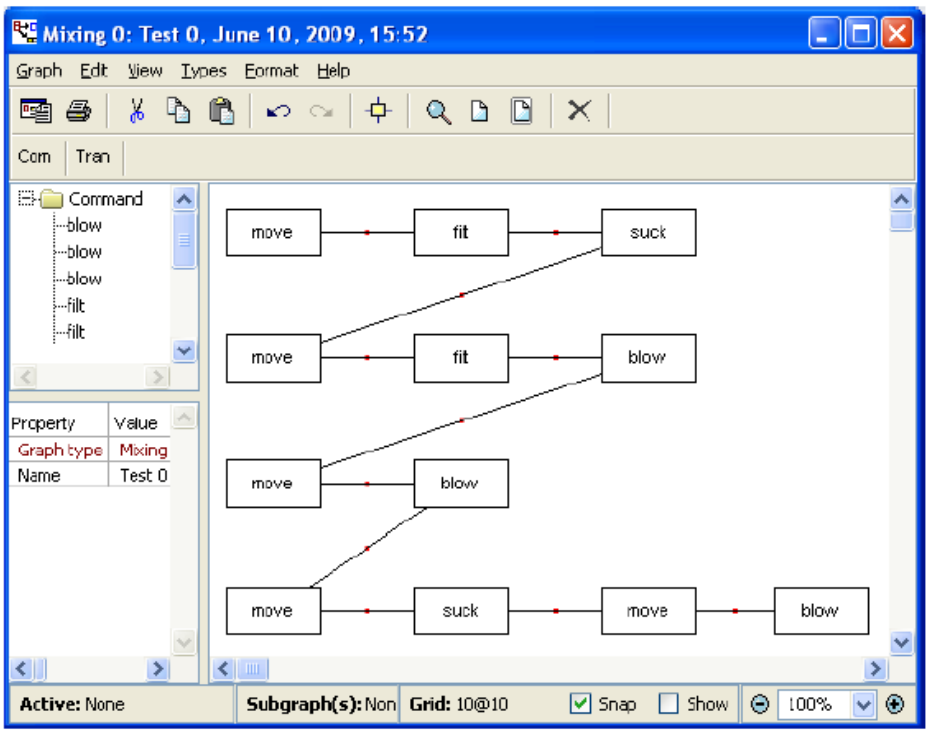
3．そのまま隣の Object（filter）へドラッグして、マウスボタンを離す。新しい Binding がこれら Object 間に作成される。



4．さらに、Object（filter）をクリックし、押した状態のままにする。

5．そのまま隣の Object（suck）へドラッグして、マウスボタンを離す。新しい Binding がこれら Object 間に作成される。

6．下図のようになるまで、上記４,５を繰り返し、全ての必要な Binding を作成する。



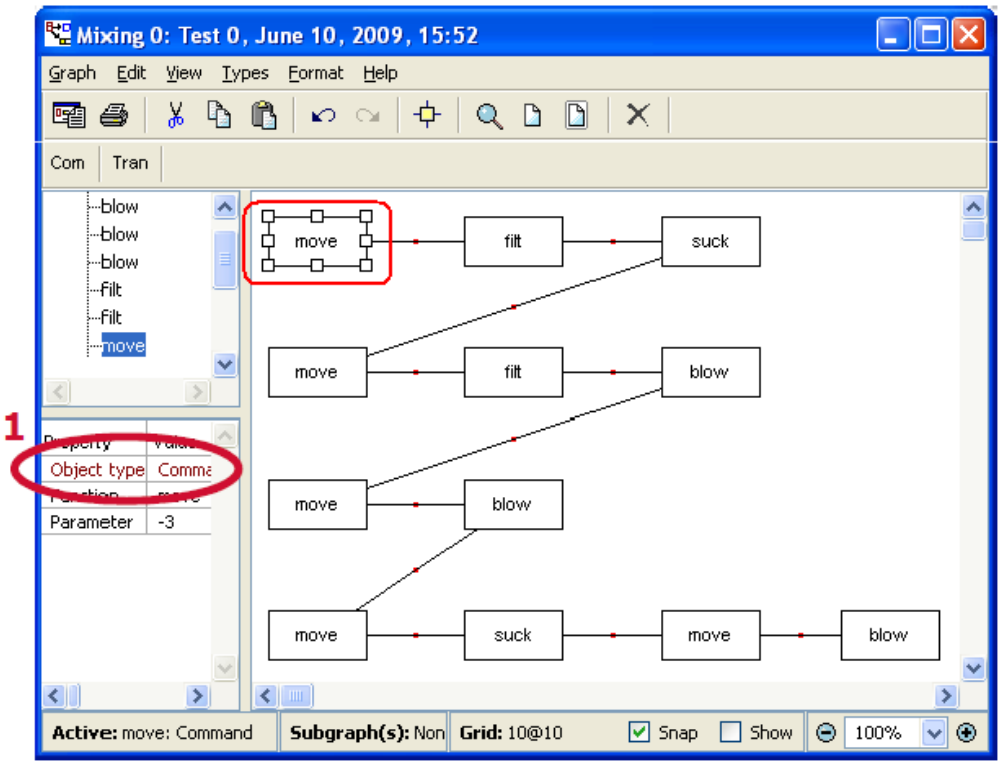
7．全 12 個の Object に対して Binding が作成できたら、描画エリアで右マウスボタンを押す。(Transition 生成終了させる)

8．作業を保存するために、MetaEdit+メインウインドウで Commit ボタンを押す。

・Object に対するシンボルの作成

1．ダイアグラムで、ある Object（ここでは、左上にある Object（move））をクリックする。

2．ダイアグラムの左にある Property シートで、shift キーを押しながら“Object type”をダブルクリックする。シンボルエディタが開く。



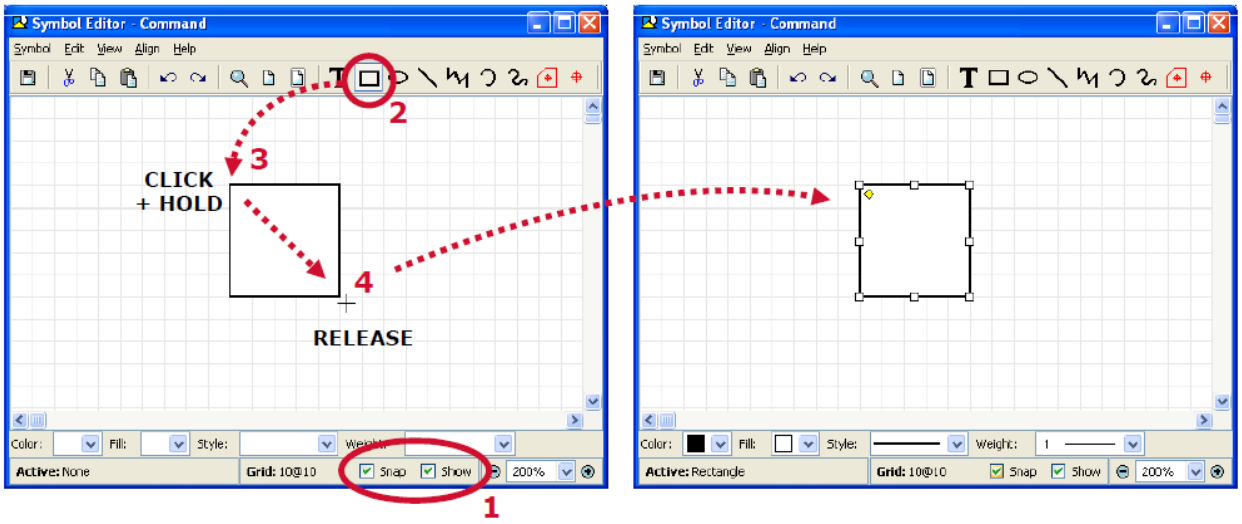
[シンボルエディタ：四角形の描画]

1．シンボルエディタの下部にあるステータスバーの“Snap”、“Show”にチェックを入れる。

2．ツールバーの“Rectangle”ボタンをクリックする。

3．下図のように、描画エリアでクリックして、そのままドラッグして四角形を作成する。

4．マウスボタンを離す。

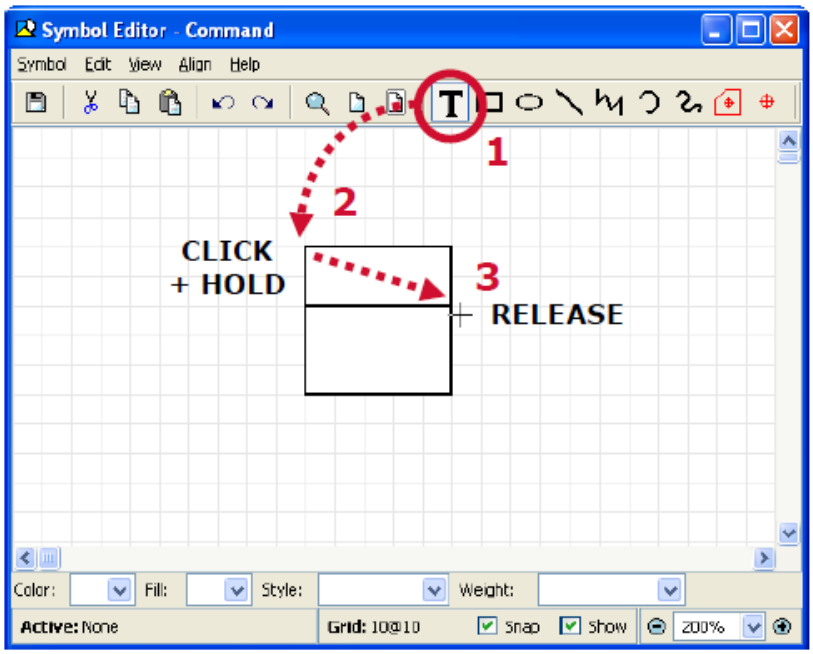


[シンボルエディタ：テキストの追加]

1．シンボルエディタのツールバーで、“Text”ボタンをクリックする。

2．下図のように、上記で作成したシンボルの左上側でクリックし、そのままドラッグしてテキストを作成する。

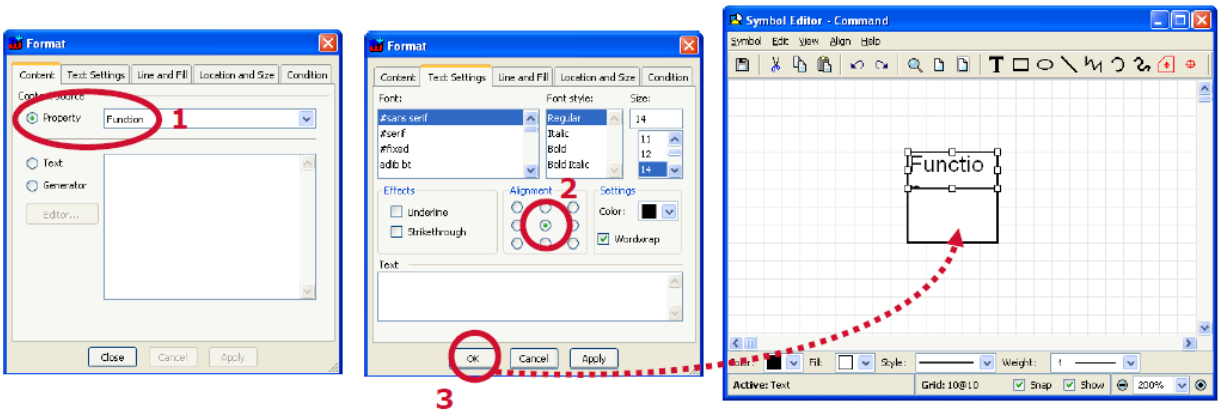
3．マウスボタンを離す。Format ダイアログが表示される。



[シンボルエディタ：テキストプロパティ]

1．Format ダイアログで、“Property” ボタンを選択し、プルダウンリストから “Function” を選択する。

2．Text Settings タブをクリックし、配置を中心に設定して、　3．OK をクリックする。



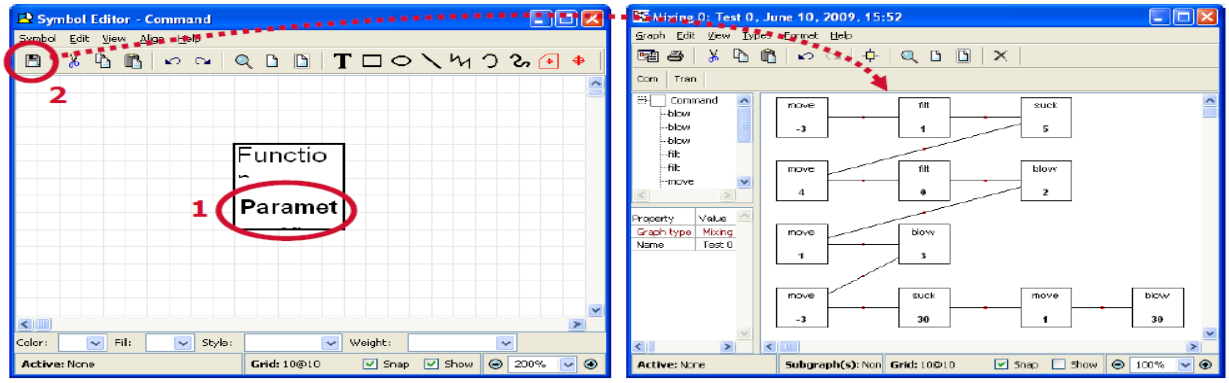
3．新しいテキストに対して、“Color（線の色）” と “Fill（塗りつぶし色）” の両方を Transparent（透明）に設定する。

[シンボルエディタ：保存]

1．上記 “Function” と同様に、“Parameter” に対してもテキストプロパティを作成する。

2．ツールバーの Save ボタンを押す。ダイアグラムエディタ上のシンボルが自動的にアップデートされる。

3．シンボルエディタを閉じる

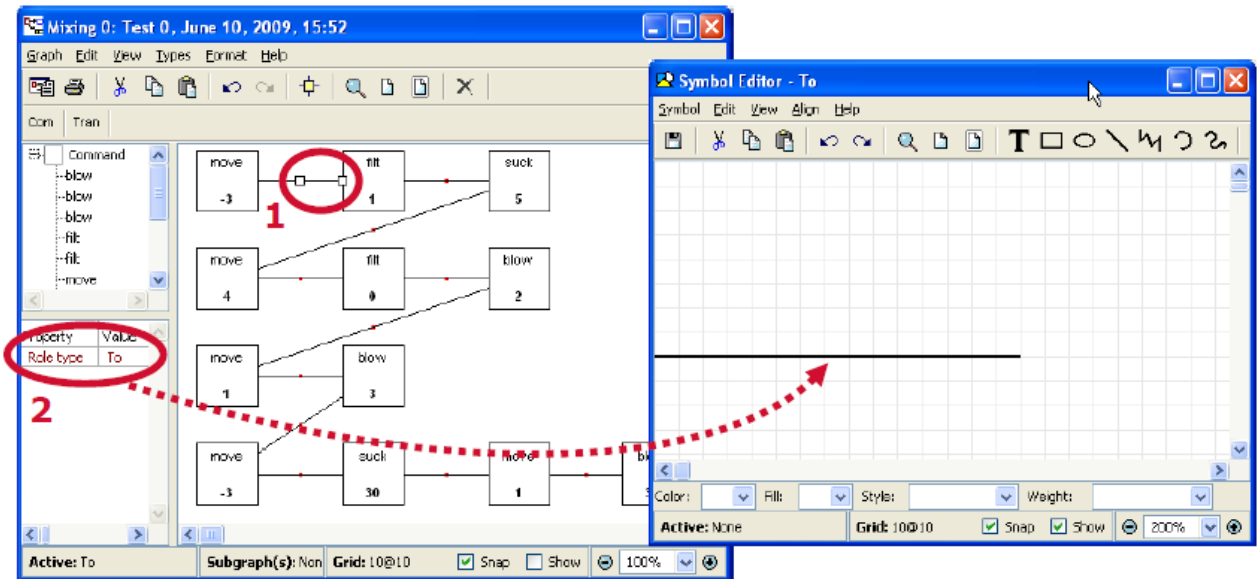


・Role に対するシンボルの作成

1．ダイアグラムで、ある “To” Role を選択（クリック）します。

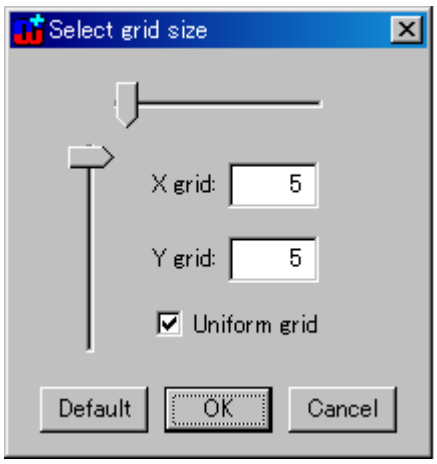
2．ダイアグラムの左にある Property シートで、shift キーを押しながら “Role type” をダブルクリックする。シンボルエディタが開く。

■シンボルエディタで Role は、オブジェクトを終端とする直線を Role 線として表記。

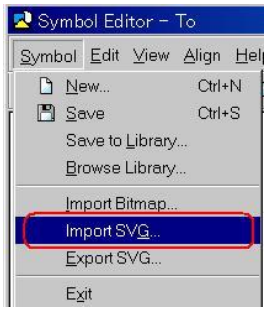


[シンボルエディタ：Role]

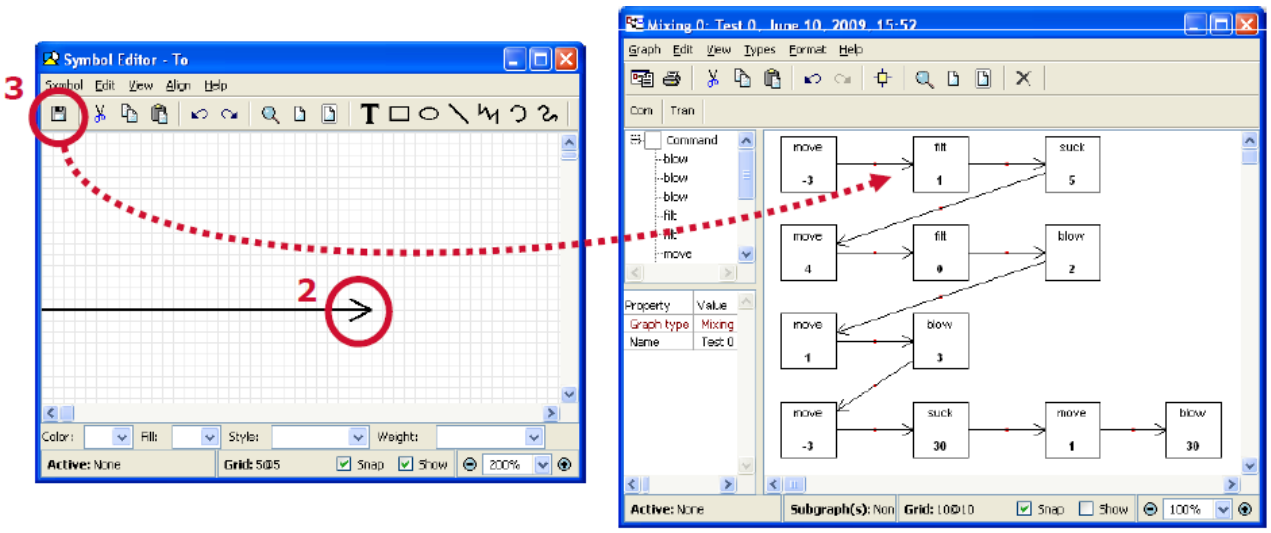
1．シンボルエディタ [View]→[Choose Grid]を選択し、グリッドとして、5×5 を設定する。



2．シンボルエディタのツールバーで、“Line”ボタンをクリックし、下記のように直線を矢印として作成する（2 マス×1 マス）。または、シンボルエディタの[Symbol]→[Import SVG]を選択し、ダイアログで“To.svg”ファイルを開く。



３．Save ボタンを押す。ダイアグラムエディタ上の Role シンボルが自動的にアップデートされる。
４．シンボルエディタを閉じる。



以上で、“Version 0 : Unityped Modelling Language　のモデリング言語（メタモデル）をコードの解析から容易に作成できることを確認し、それを用いてアプリケーションをモデリングしました。
しかしながら、これではコマンド言語（ＤＳＬ）レベルのモデル言語であり、コードレベルの抽象度であるため、生産性の向上や、再利用の促進、高い品質の確保は望めません。

MetaEdit+ メタモデリング機能

■ ダイアグラムエディタは、標準的なエディット機能を提供
- Undo/redo（アンドゥー/リドゥー）
- カット/コピー/ペースト/専用のペースト
- リファクタリングのためのリプレース
- トレースのための情報
- グリッドの操作：スナップ、アラインメントなど。
- インポート/エクスポート、自動配置

■ ジェネレーターとの統合
- コードジェネレーター
- ドキュメントジェネレーター
- モデルチェックと解析
- メトリクス の生成など。

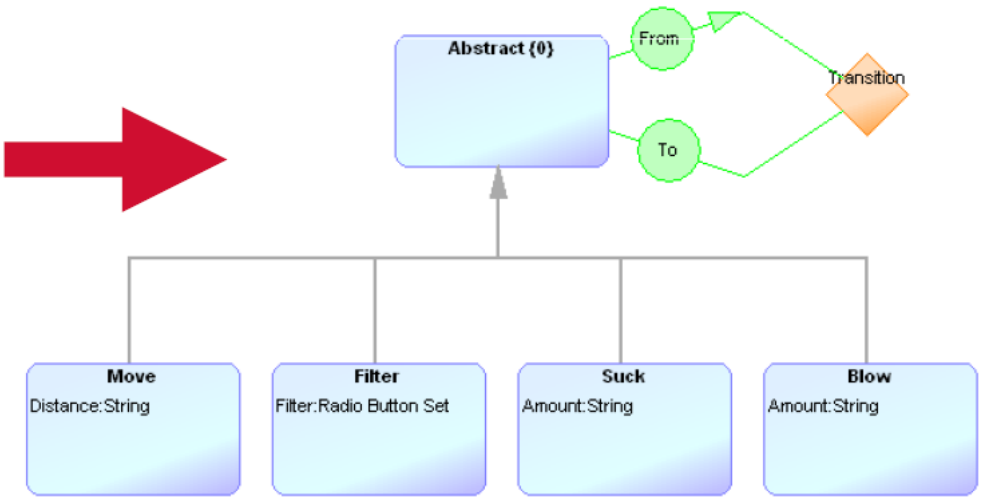
■ ブラウジング、インポート、エクスポート、プログラマブルAPIなどを介して、他のツールと連動

＜Version 1：Domain-specific concepts＞　ドメインスペシフィックな コンセプトを加える

■ コマンド（move、filt、suck、blow）ごとに独自のタイプを作成
　ーオブジェクト指向アプローチ 例：コマンドパターン

■ 以下のような、新しいメタモデル（Mixing 1）を作成する。

```
move(-3);
filt(1);
suck(5);
move(4);
filt(0);
blow(2);
move(1);
blow(3);
move(-3);
suck(30);
move(1);
blow(30);
```

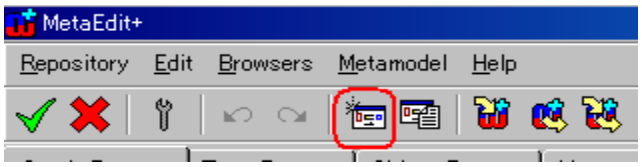


[各タイプのパラメータを設定]

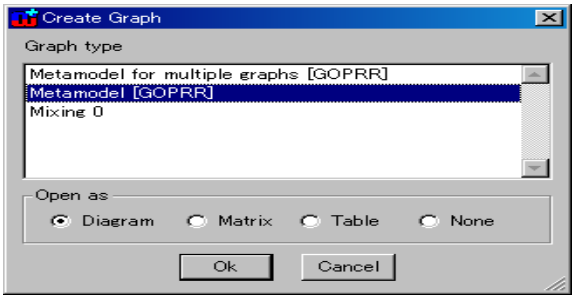
■ タイプ毎に、異なるパラメータリストを持つことを留意する
– 例：filter値は、move値と異なる。
– Filter {0,1,2}, move -10..10, suck/blow 0..30　Filter{0,1,2}、move{－10～10}、suck/blow{0～30}

[2 つ目のメタモデルの作成準備]

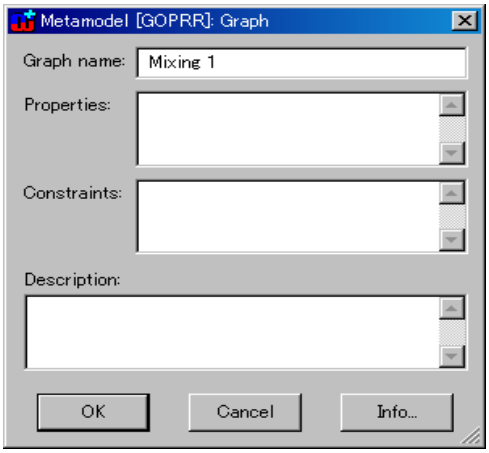
1．MetaEdit+メインウインドウで、ツールバーの“Create Graph”ボタンを押す。



2．Create Graph ダイアログで、“Metamodel [GOPRR]”を選択して OK をクリックする。



3．グラフ名として、“Mixing 1”を入力する。

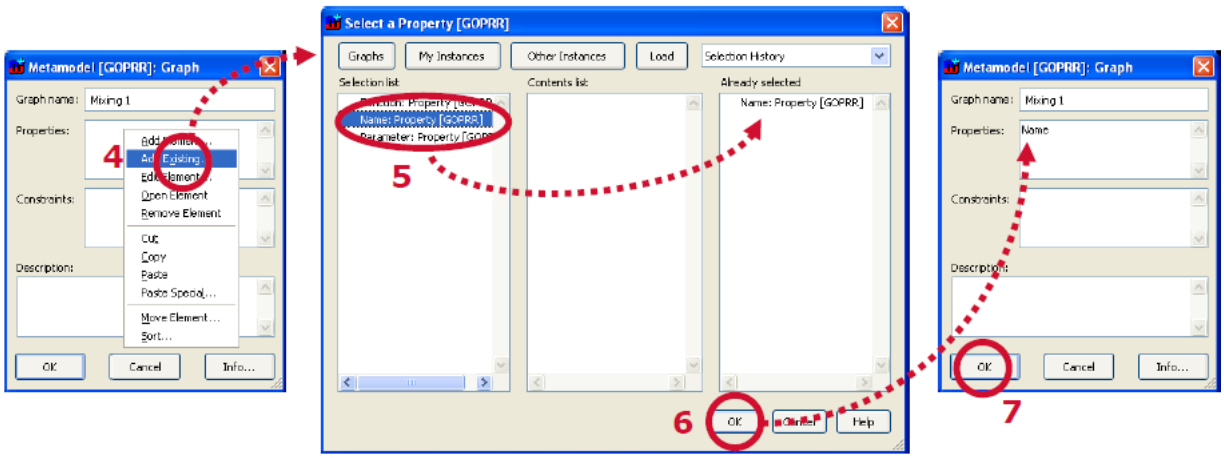


4．既存の Name プロパティを再利用する。
Properties リストで右クリックし、“Add Existing”を選択する。

5．Selection list で、“Name”プロパティをダブルクリックする。“Name”が、Already selected リストに表示される。

6．OK をクリックすると、“Name”が新しいグラフのプロパティとして再利用される。

7．OK をクリックする。



・アブストラクト（抽象）オブジェクト（Object）の作成

1．ヒント：グリッドを 50×75 と設定しましょう。

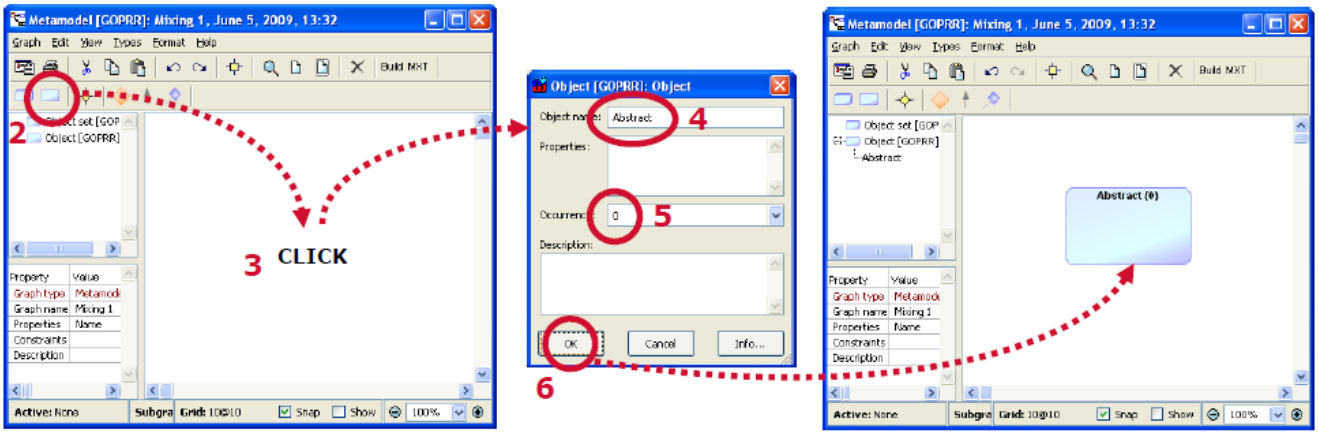
2．タイプツールバーで Object をクリックする。

3．描画エリアでクリックする。Object 定義ダイアログが開く。

4．Object の名前として、“Abstract”を入力する。

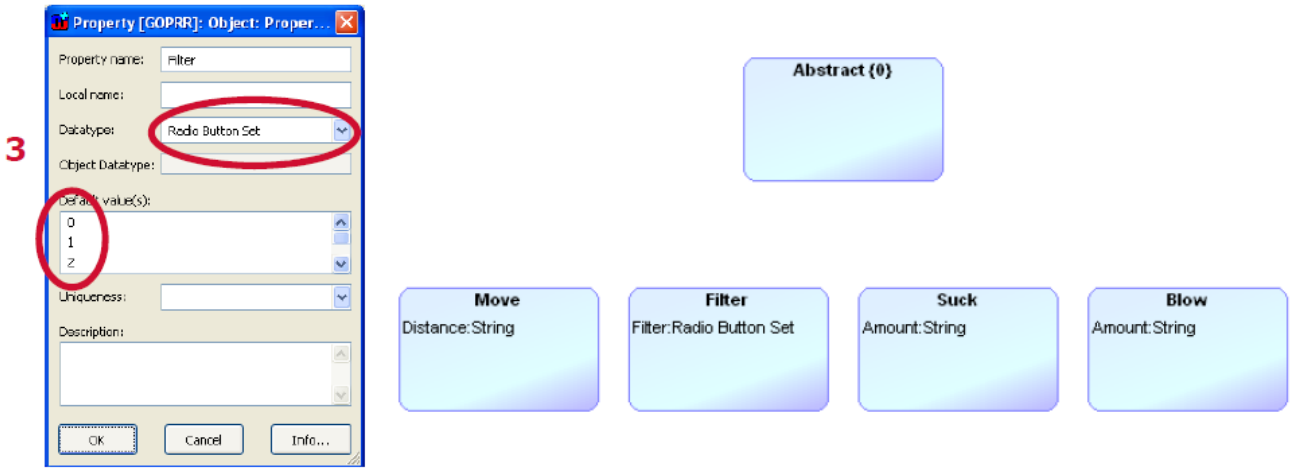
5．Occurrence に“0”を設定する。（アブストラクト（抽象）オブジェクトとして）

6．OK をクリックする。新しい Abstract の Object が作成される。



さらに、上記と同様の方法で、下図のように他のオブジェクトも作成し、プロパティを定義します。

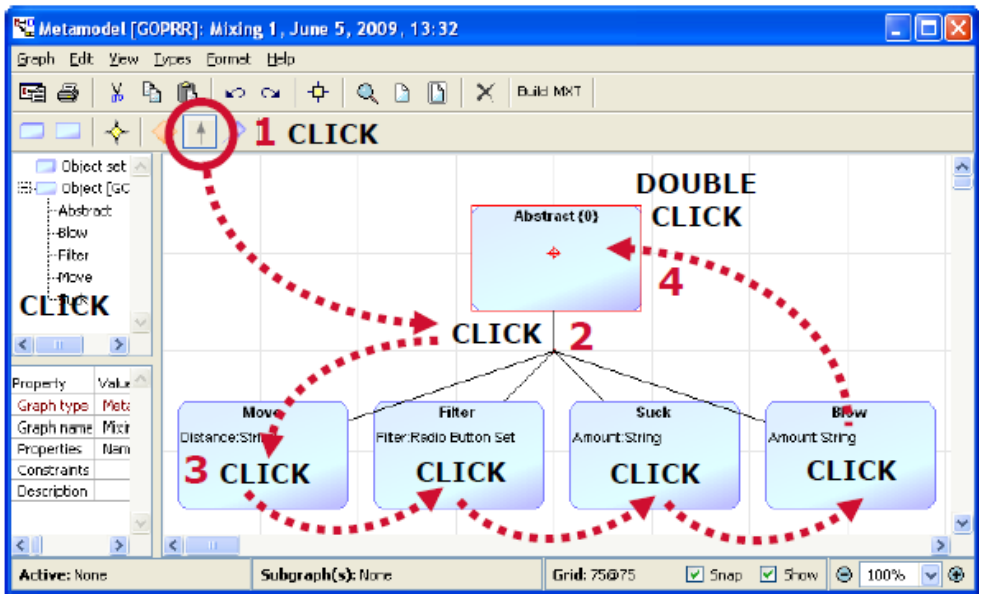
1．Move オブジェクトを作成し、プロパティとして“Distance”を定義する。
2．Filter オブジェクトを作成し、プロパティとして“Filter”を定義する。
プロパティの Datatype として“Radio Button Set”を選択し、Default Value として“0,1,2”を縦に入力します。
3．Suck オブジェクトを作成し、プロパティとして“Amount”を定義する。
4．Blow オブジェクトを作成し、プロパティとして上記で作成した“Amount”を再利用して定義する。
（Add Existing…）



・Inheritance（継承）を使用
■ Move、Filter、Suck、Blow は、Abstract の派生です。そこで、Inheritance リレーションシップを使用します。
■ N-ary バインディングは、2 つ以上のロールを持つバインディング

1．タイプツールバーの Inheritance をクリックする。
2．Inheritance の接合ポイントを配置するために、描画エリアの Abstract の下あたりでクリックする。
3．そして、オブジェクトを接続するために、Move、Filter、Suck、Blow をクリックして派生とする。

4．Abstract オブジェクトでダブルクリックし、Inheritance を終了。



2 つ目のメタモデルを完成させましょう。

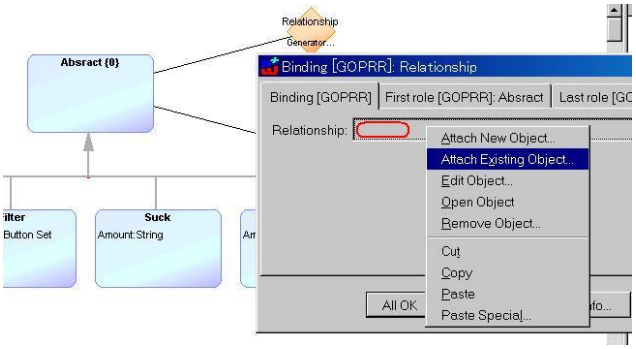
1．継承の n-ary のバインディングをビジュアル化して完成する

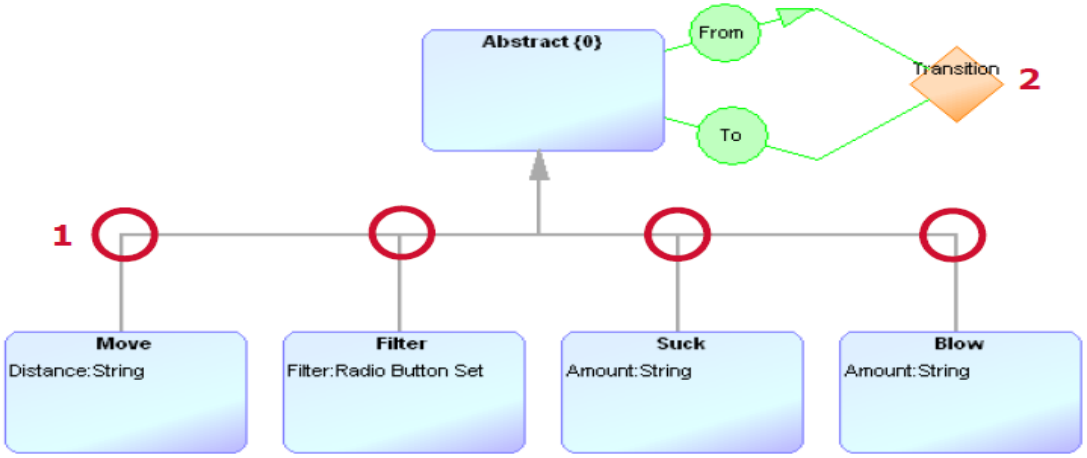
－下図のように、各 Role の線をクリック・ドラッグして折点を設ける。

2．Transition バインディングを追加する。（Version 0 の時のように）

－既存の Transition リレーションシップ、From、To ロールを再利用しましょう。

Binding 作成時ダイアログで右クリックし Add Existing Object から既存 Transision を選択します。First role タブ、Last role タブ内もそれぞれ Add Existing Object から既存の From, To ロールを選択します。





■ 新しいメタモデルを生成するために、“Build”を押す。

■ 新しく作成したMixing 1を使って、Test 1グラフを作成しましょう。

– グリッド100×100で、下記モデル構造を作成します。

– 下図、赤字の move などはなく、デフォルトのシンボルだけが見えます。

[Test 1 グラフの作成]

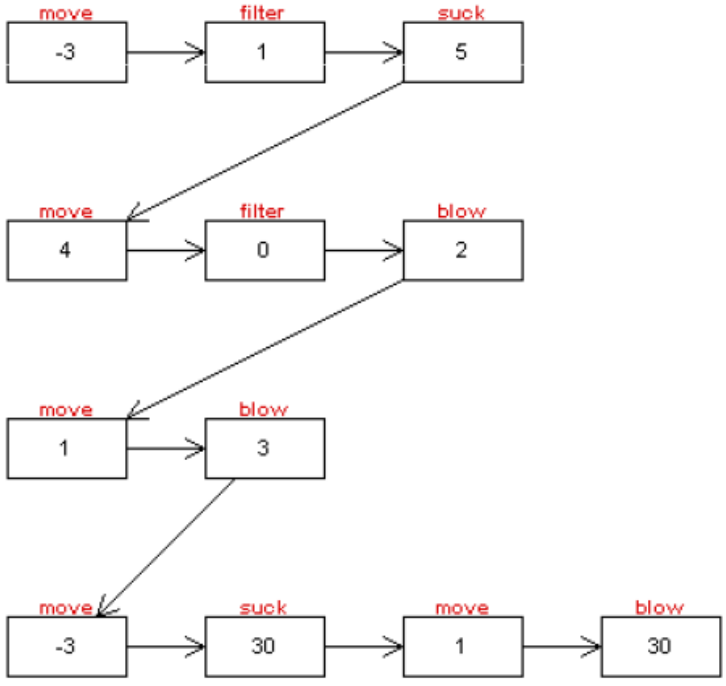
1．MetaEdit+メインウインドウで、ツールバーの Create Graph ボタンを押す。

2．Create Graph ダイアログから Graph タイプとして“Mixing 1”を選択し、OK をクリックする。

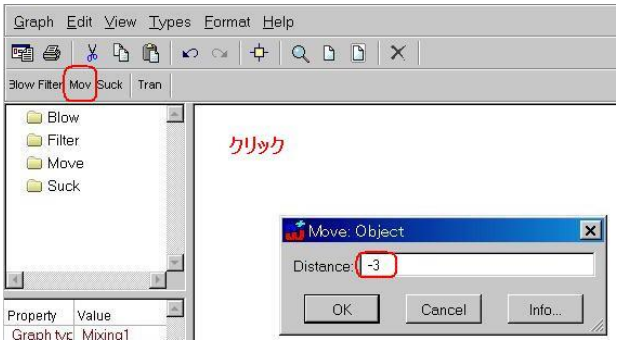
3．新しいグラフの名前を入力するダイアログが開くので、“Test 1”を入力する。

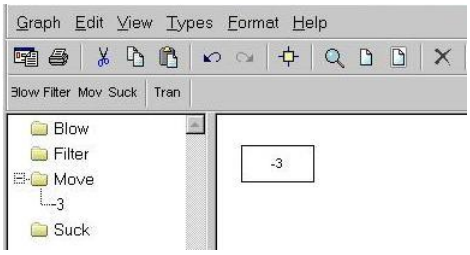
4．OK をクリックすると、ダイアグラムエディタが開く。

下図のようなモデルを作成する。シンボルだけが見え、下図の赤字は表示されないので注意する。

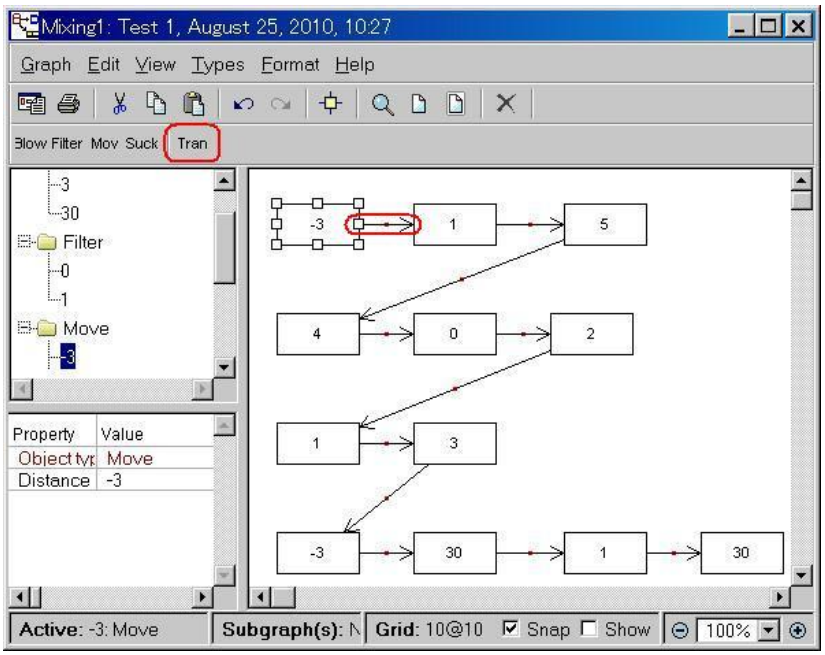


・ダイヤグラムエディタ内で Move を選択後、描画エリアでクリックし, Distance に－3 を入力



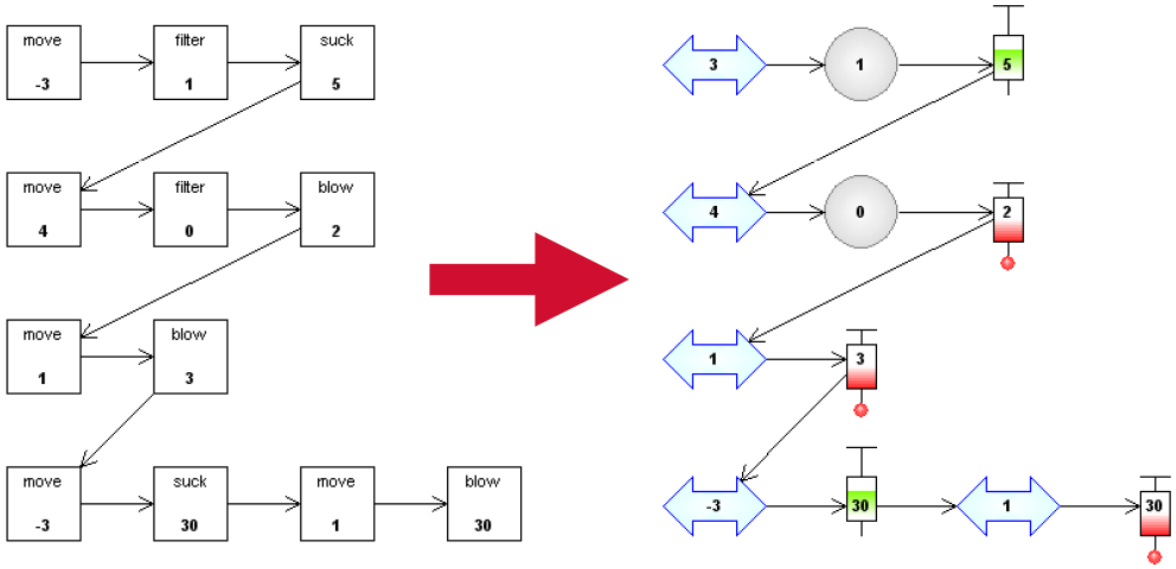


・残りのシンボルを追加し、各シンボルを Transition を使い下のモデルを完成させて下さい。



[シンボルの作成]

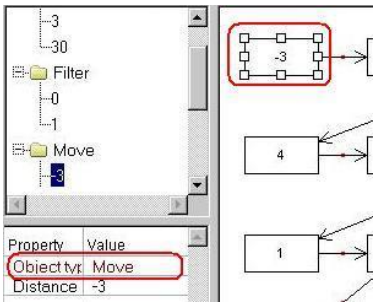
各 Object の意図を明確にするために、例えば、下記のような形、色のシンボルを作成してみましょう（“Test 0” で作成したことを参考に）。もしくは、move、filter、suck、blow、に関して、“Move.svg”、“Filter.svg”、“Suck.svg”、“Blow.svg” を使用しましょう。



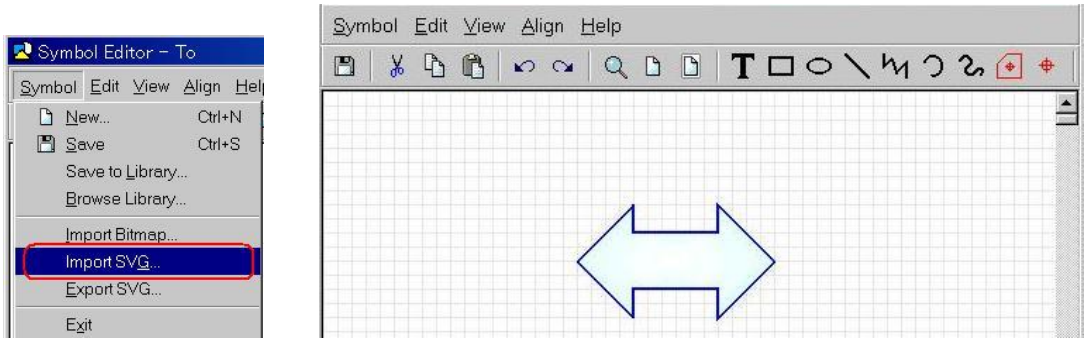
＊move を例にとったシンボル作成手順

1．ダイアグラムで、“move” を選択（クリック）します。

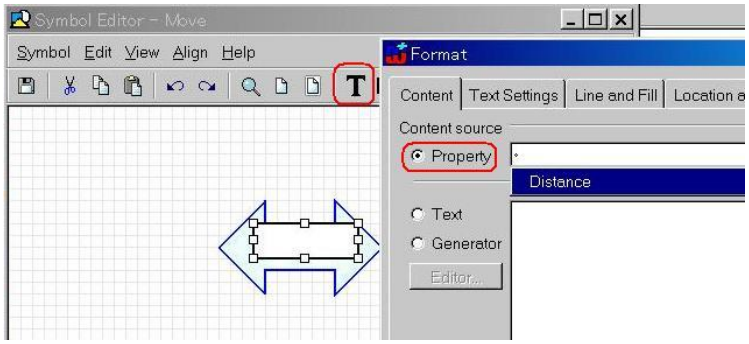
2．ダイアグラムの左にある Property シートで、shift キーを押しながら “Object type” をダブルクリックします。シンボルエディタが開く



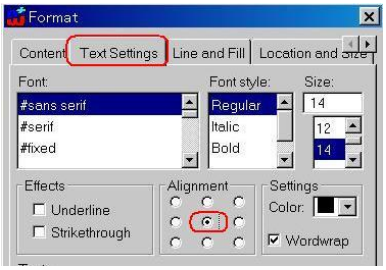
3．シンボルエディターにて 既存のシンボルデータ move.svg をインポートします。



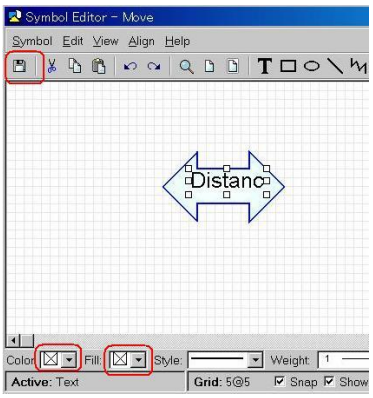
4．インポートされた move シンボル内に Text： Distance を設定します。



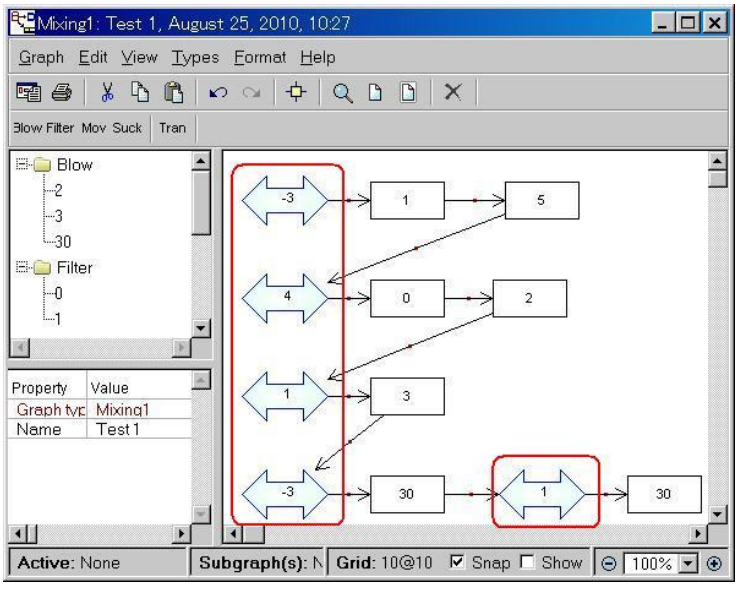
＊ Text Settings タブを利用し、Text の位置を設定します。



5．Color, Fill 共々 Transparent に指定し、 Save ボタンを押します。



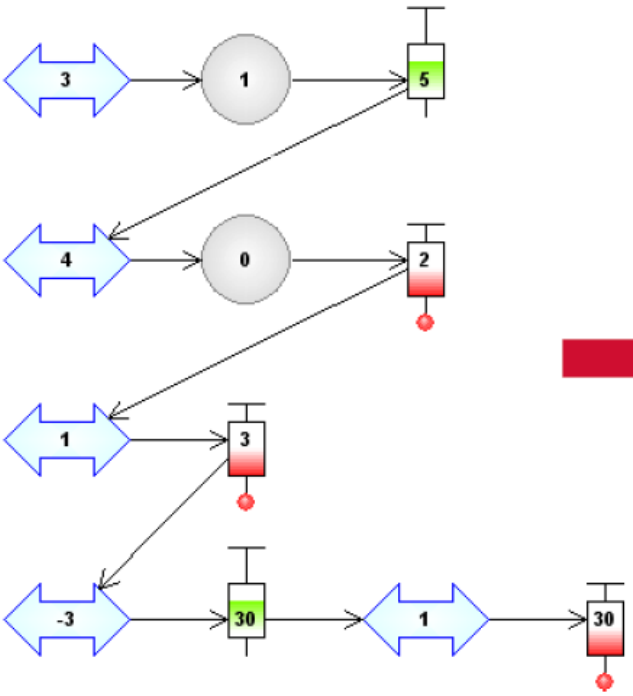
6．move が新しく設定した シンボルに置き換わっていることを確認して下さい。



[コードジェネレーター の設定]

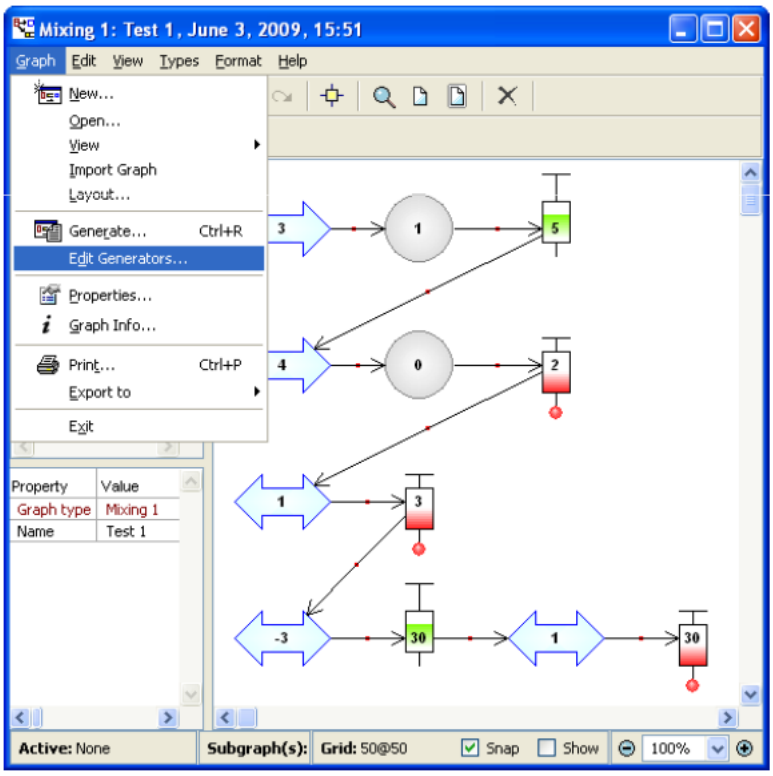
■ Objectに関してコマンドを書き出す。

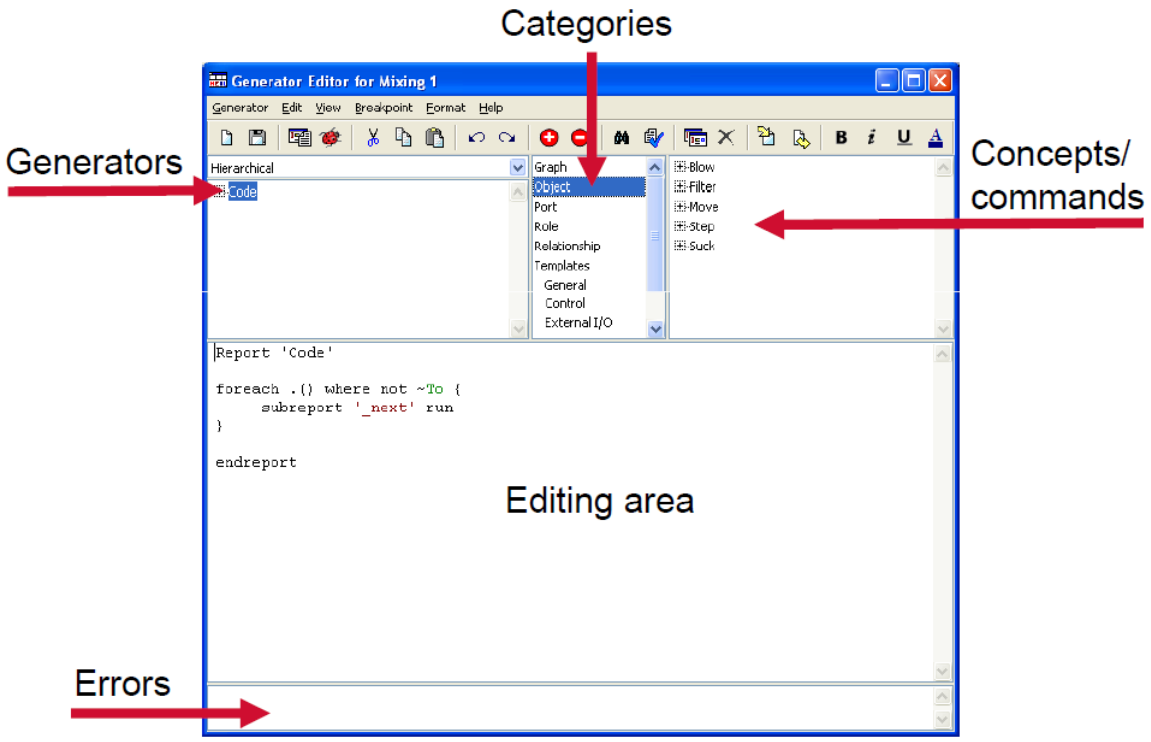
■ 次の Object への接続を見つけるために、リレーションシップの方へ進む（これを繰り返す）。そして、下記のようなコマンドを出力。




```
move(-3);
filt(1);
suck(5);
move(4);
filt(0);
blow(2);
move(1);
blow(3);
move(-3);
suck(30);
move(1);
blow(30);
```

■ ダイアグラムエディタで、メニュー“Graph->Edit Generators”を選択、Generator Editorを開く。



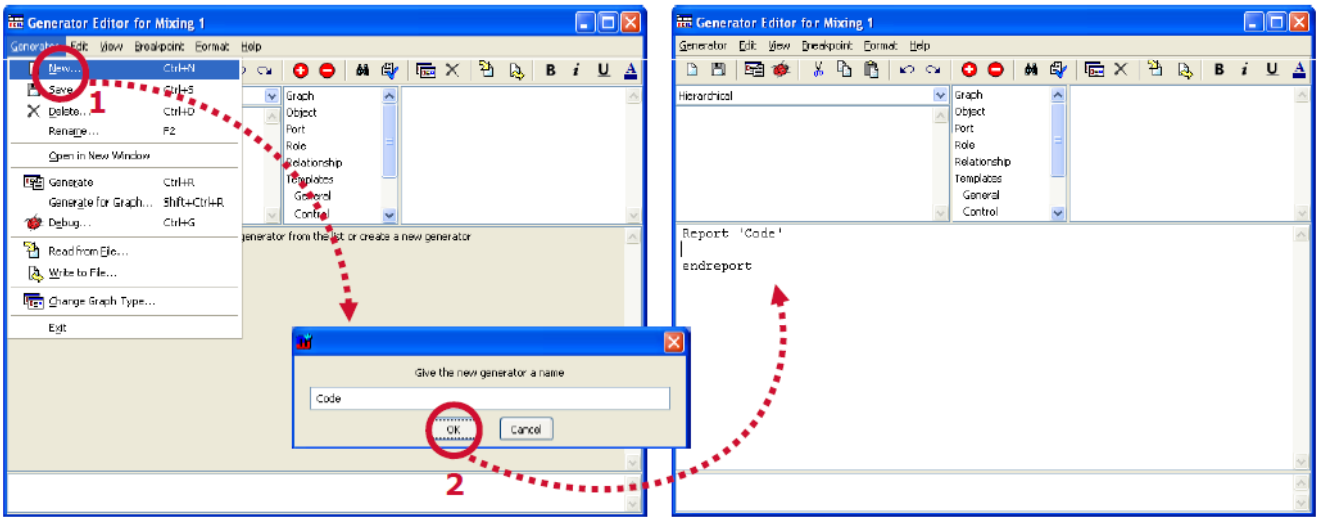


＜ジェネレータ用のエディタ画面＞

・新しいジェネレーターの作成

1．メニュー“Generator->New”を選択する。

2．新しいジェネレーターの名前として“Code”を入力する。すると、編集エリアに新しい空のジェネレータースクリプトが表れます。

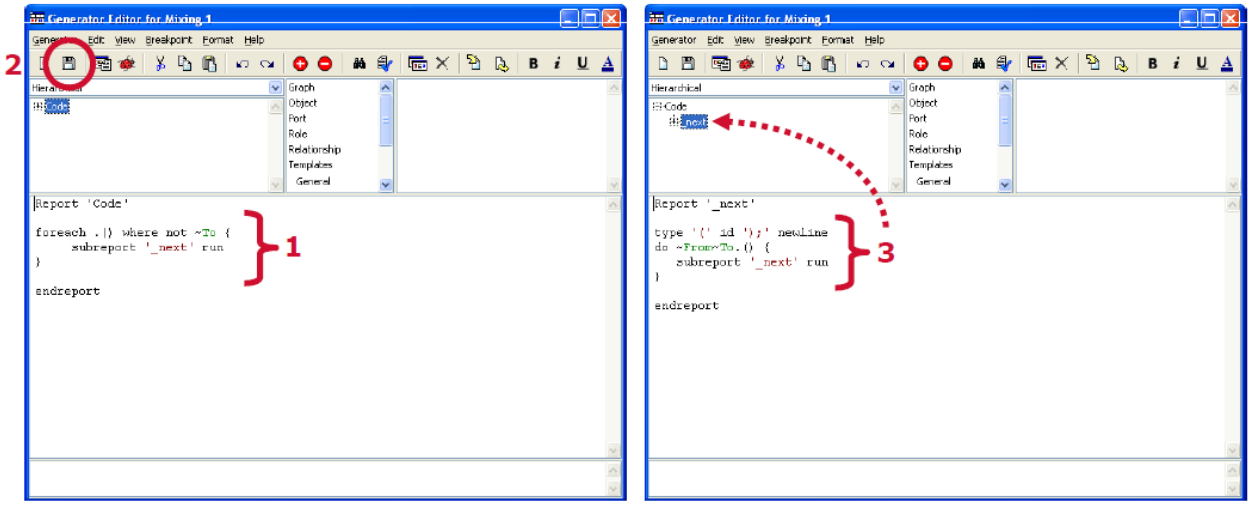


・ジェネレータースクリプトの作成

1．編集エリアでジェネレーター定義を記述する。（スクリプトの内容は、下記参照）

2．ツールバーの Save を押すと、ジェネレーターリストに“Code”が現れる。

3．同様に、“_next”と呼ばれる別のジェネレーターを作成する。（スクリプトの内容は、下記参照）



[ジェネレータースクリプト：Code]

```
Report 'Code'
foreach .() where not ~To {
subreport '_next' run
}
endreport
```

[ジェネレータースクリプト：_next]

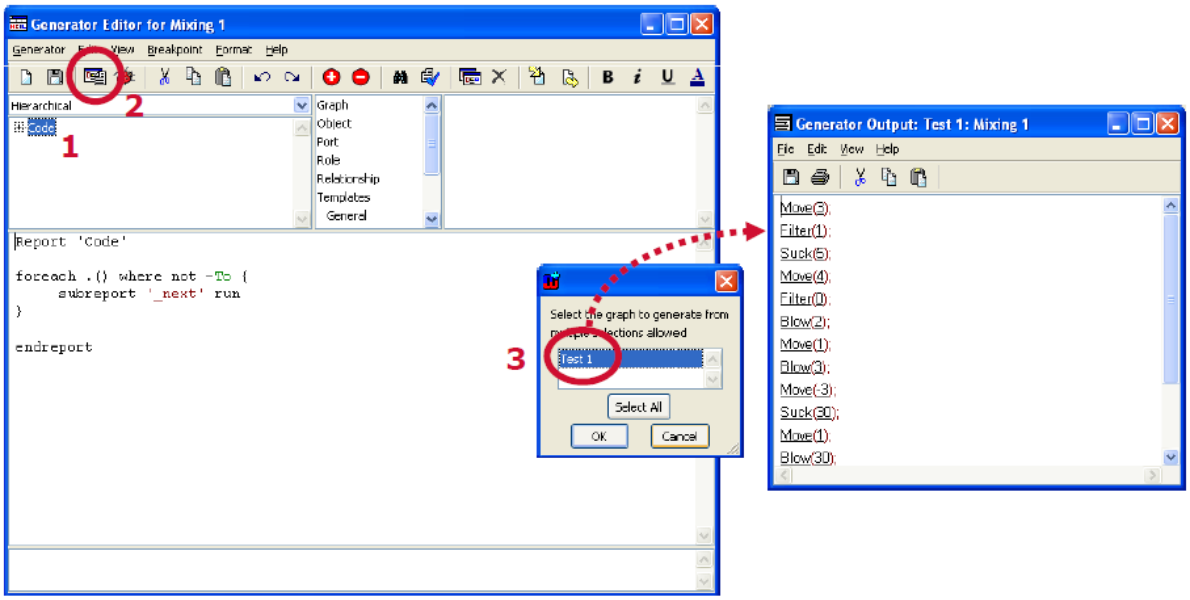
```
Report '_next'
type '(' id ');' newline
do ~From~To.() {
subreport '_next' run
}
endreport
```

・ジェネレーターのテスト

1．ジェネレーターリストから“Code”を選択する。

2．ツールバーで“Generate”を押す。

3．リストから、“Test 1”を選択すると、Gnerator Output ウインドウが開き、ジェネレーターの出力が表示される。



[ジェネレータースクリプトの説明]

ジェネレーターは、独自のスクリプトを使ってグラフ内（ここでは Test 1）を巡回し、見つかったグラフ内の情報（オブジェクトのプロパティ値など）を取り出し、コードとして出力します。IF などのコマンド命令を使って、プロパティの値に応じて出力内容を変更することも可能です。また、あるスクリプト内で、別に定義したスクリプト（サブジェネレーター）を実行することも可能です。

下記スクリプト内の‘黒字’はスクリプトコード、‘赤字’は実際に出力されるコードです（''で括られた部分は、そのままコードとして出力されます）。

・“Code”ジェネレータースクリプト

```
Report 'Code'
foreach .() where not ~To {     // Toロールを持たないオブジェクトを探索
                                //  （つまり、最初のコマンド（move）から開始）

  subreport '_next' run         //  ‘_next’サブジェネレーターを実行
}
endreport
```

・“_next”ジェネレータースクリプト

```
Report '_next'
type '(' id ');' newline        // 現在のオブジェクトのタイプ（例：move）、
                                //  id（最初のプロパティ値）、改行を出力
```

```
                                    //  （例：“move(-3);”）
do ~From~To.() {                    //  順番に次のオブジェクトへ向かうために、Fromや
                                    //  Toロールを探索


  subreport '_next' run             //  再帰的に‘_next’サブジェネレーターを実行
}
endreport
```

上記スクリプトにより、下記が出力されます。

```
move(-3);
filt(1);
suck(5);
move(4);
filt(0);
blow(2);
move(1);
blow(3);
move(-3);
suck(30);
move(1);
blow(30);
```

以上 Version 1 では、ドメインスペシフィックなコンセプトをメタモデルに設定すること、コードジェネレータを容易に作成する仕組みを紹介しました。しかしながらこのレベルでは、まだ抽象度は低く、また再利用性は良くありません。

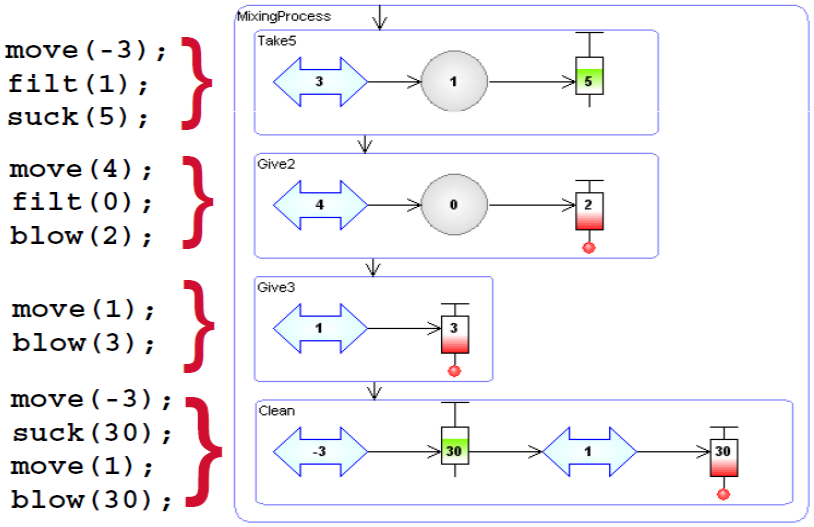
＜Version 2：論理的にグループに分割する＞

■ コード内から論理的なグループを抽出し、視覚的なブロック（あるいは固まり）として反映させる

– ボックスで囲んで、まとめる（コメントで仕切られるコード領域など）

– ボックス＋リレーションシップ(cf. goto) で構成

下図のようにコマンドをグループ化し、それぞれのグループをリレーションシップで接続する。



・Step オブジェクトを、Version 1 のグラフィカル GOPPR モデルに追加する

■サブタイプ（特殊型）としてStepオブジェクトを作成し、メタモデルを更新する

（Mixing 1：Metamodel [GOPRR]）

■Stepオブジェクトに“Name”プロパティを作成する（“Add Existing”を使用する）

■ Inheritanceリレーションシップにstepオブジェクトを加える

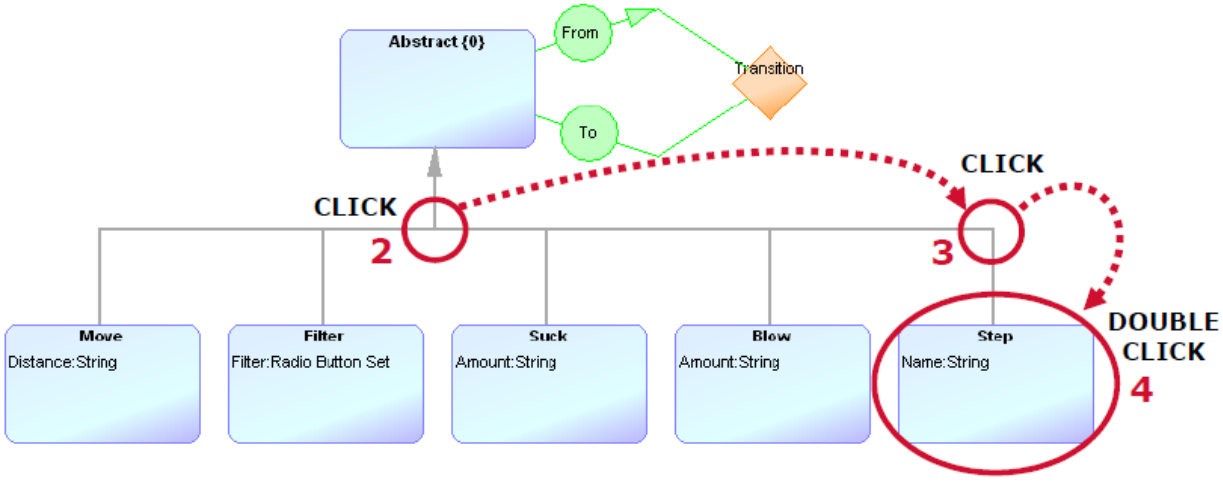
1．ツールバーで、Inheritanceをクリックする。

2．既存のInheritanceリレーションシップの接続点をクリックする。

3．折点としてクリックする。

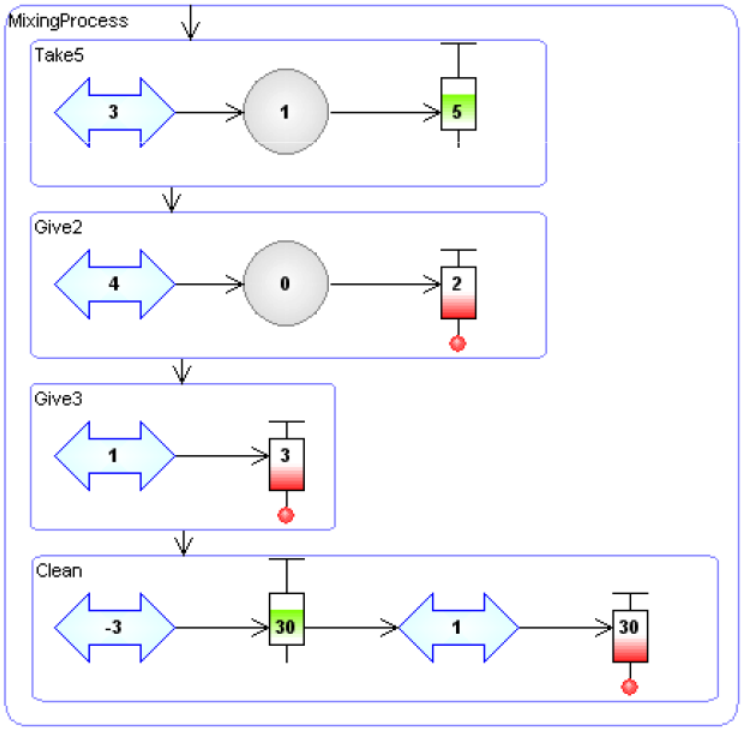
4．最後に、Stepオブジェクト上でダブルクリックする。

■ メタモデルを更新するために Build を押す。



・Step を用いて、アプリケーションモデルをリファクタリング（グループ化する）

下図のようなグラフを作成する。(Version1 の“Test 1”を改良)。各 Step の名前については、下図を参照。



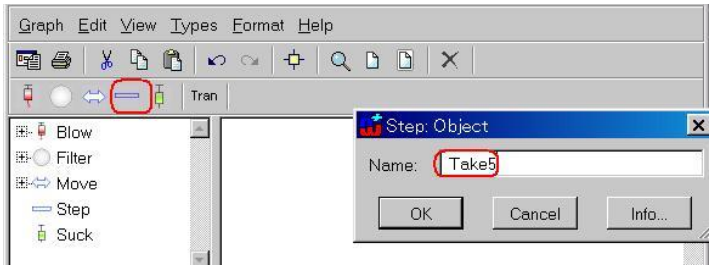
■ Stepオブジェクトを追加する

■ シンボルを編集するために、ObjectタイプでShiftキーを押しながらダブルクリックする。

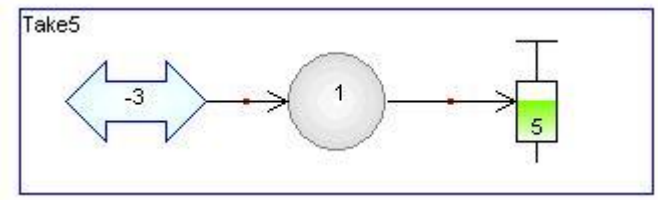
– Rectangleで長方形を作成する（背景を透明にして、左上に小さなテキスト（Name）を作成する）

■ 全てのグループに対してStepを追加する。

stepオブジェクト追加後はオブジェクト選択し描画エリアでクリックしName: Take5を入力します。



– 下図のように、Stepのサイズを変更して囲む。



■ TransitionでStepを接続する。

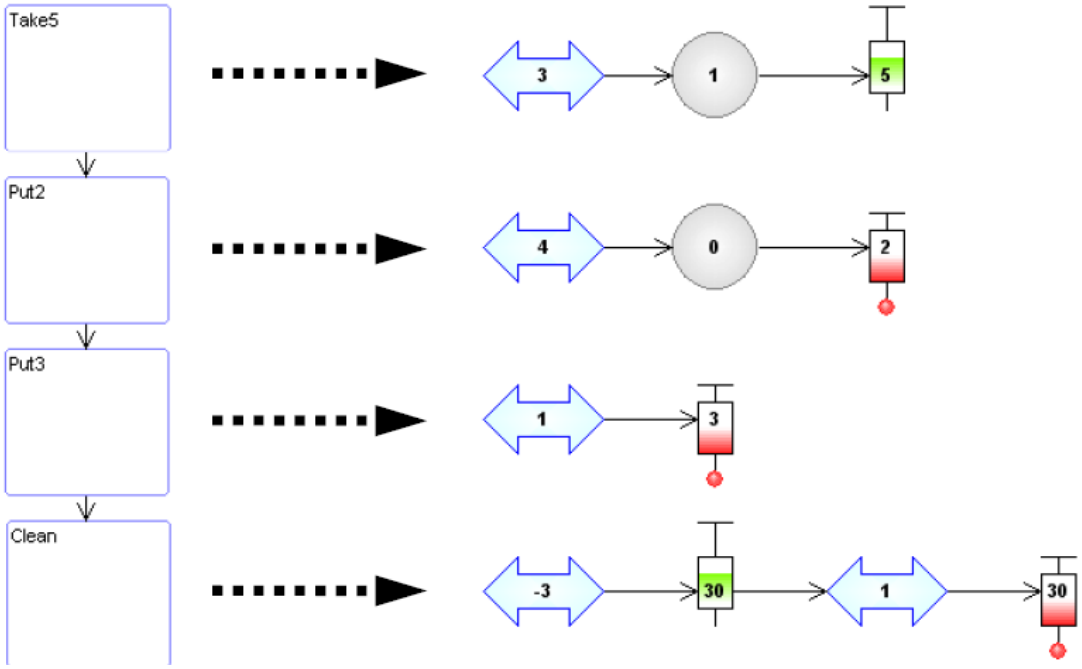
■ 元々あった Transition は削除する。

以上 Virsion２では、Virsion１で作ったメタモデルを修正し、それを既存のアプリモデルに反映させる方法を確認しました。次に、この変更を用いて、再利用性、抽象度を高める方法をご紹介します。

＜Version 3：モデルの再利用＞

■ Virsion 2 で追加したStepオブジェクトを用いて、モデルの階層化を可能にする

– モデルの再利用の支援



・サブグラフの作成

■ 階層構造のモデリングに対応するメタモデルを作成

– MetaEdit+メインウインドウの“Create Graph”をクリック

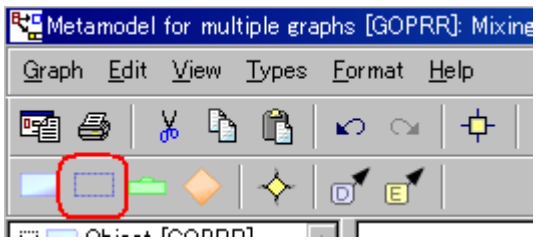
– リストから“Metamodel for multiple graph [GOPRR]”を選択

（multiple graphで、追加メタモデルを統合することに注意）

– 同じメタモデルの名の“Mixing 1”を入力する。

■ ダイアグラムにGraphを加える

– ツールバーで“Graph”ボタンをクリックする。



– 描画エリアでクリックする。

– 大き目のGraphボックスにする。

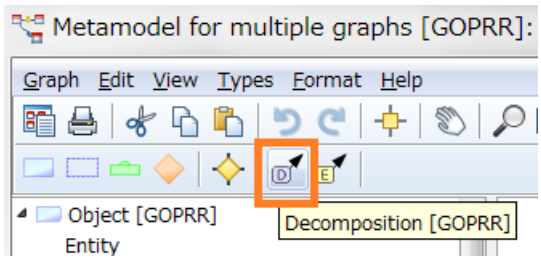
– Graphボックスで右クリックし、“Decomposition”を選択する。
(ver5 以降では、“Open Subgraphs”に変更されています)

– 現在のメタモデル“Mixing 1”を選択し、OKを押す。

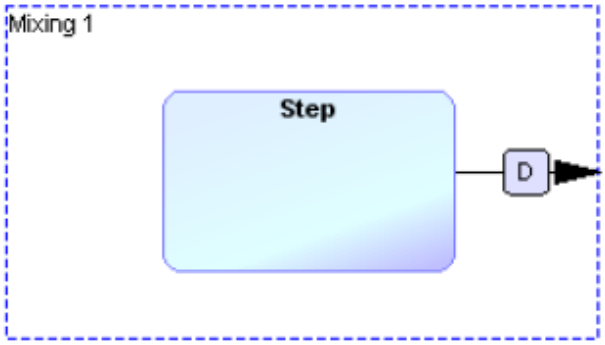
■ モデル階層を加える

– 既存のメタモデルからStepをコピーし、新しく作成した“Mixing 1”グラフ内に貼り付ける。

– “Step”から“Mixing 1”へDecompositionリレーションシップを引く。（Decompositionボタン）



– “Mixing 1”グラフ内のStepは、別の“Mixing 1”グラフへ分解される。



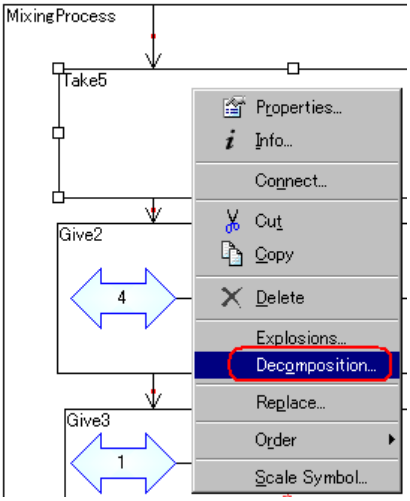
■ 新しいメタモデルを生成するために、Build を押す。

■ アプリモデル“Test 1”へ戻りましょう。

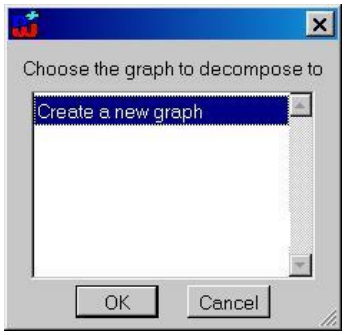
■ 各Stepに対して、モデル階層にリファクタリングする

– Step内のオブジェクトを選択し、それらを切り取る。

– Stepを選択し、右クリックで“Open Subgraphs”を選択する。

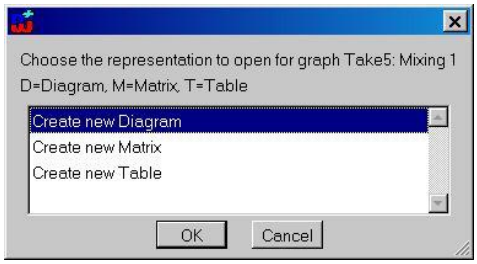


– “Create a new graph”を選択する。

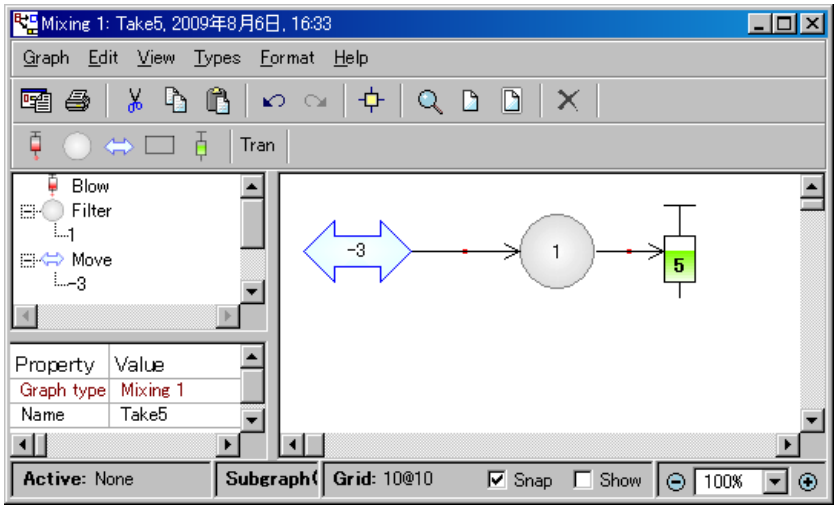


– グラフ名としてStepの名前（例えば “Take5”）を入力し、OKを押す。

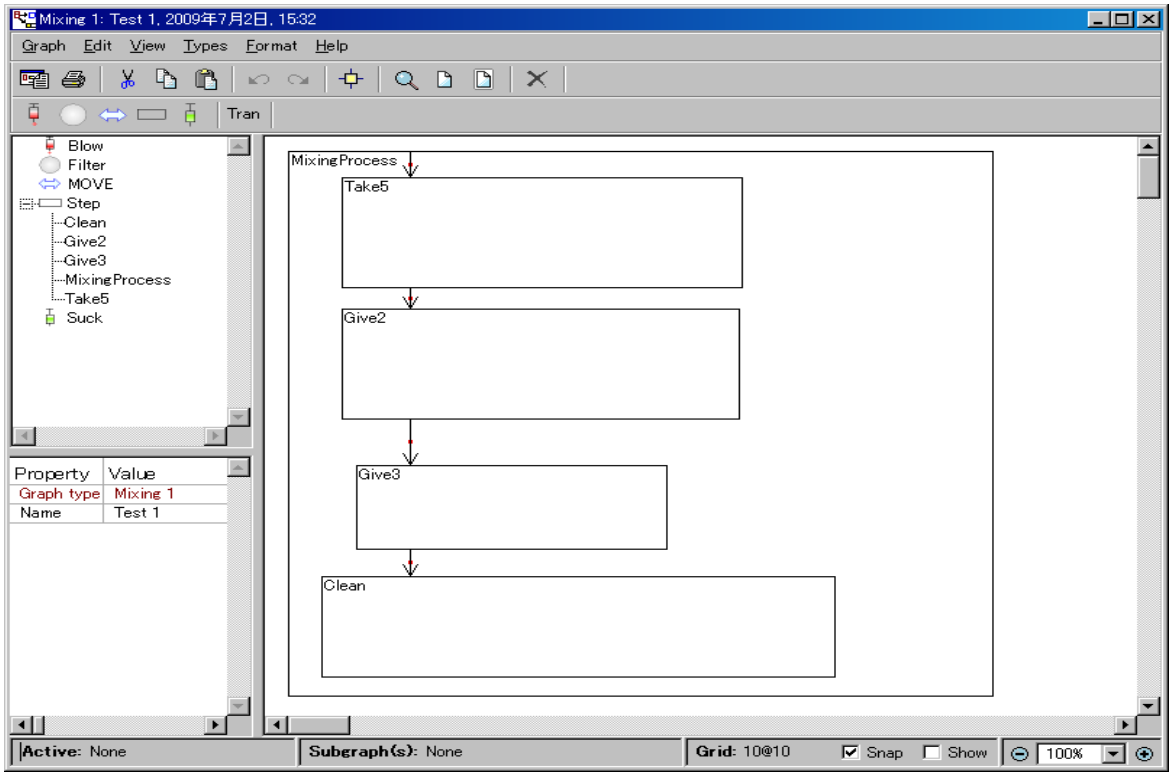
– “Create new Diagram”を選択する。



– 切り取った内容をグラフに貼り付ける。



これを全ての Step で実行すると、以下のようになります。



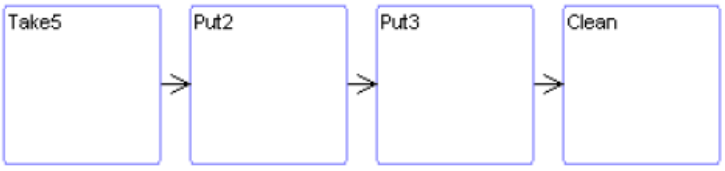
Version3 では、Version2 で抽出した、いくつかの詳細な処理をサブシステムとしてまとめて、別モデルとして管理することで、抽象度を上げることに成功し、再利用性が高まりました。
ここで、MetaEdit+ によるモデルの階層化、マルチ言語対応の仕組みを上手く活用しました。

＜Version 4：よりハイレベルなドメインコンセプト＞

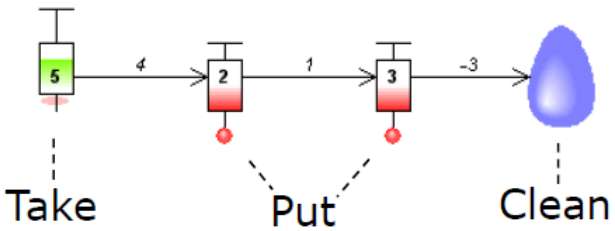
■再利用可能な固まりをタイプに設定

– 再利用をパラメータ化するために、タイプのプロパティを定義する

■Version 3:から、



■Version 4:へ



■注：Version 3 では 1 オブジェクトと設定していた Move コマンドは、Version 4 では、操作間のリレ

ーションシップの新しいプロパティとして取り扱う

・新しいメタモデルを作成する

■ ツールバーで、Create Graphボタンをクリック。Create Graphダイアログが開くので、リストから“Metamodel[GOPRR]”を選択し、OKする。Graph nameとして、“Mixing 4”を入力する。

■ 下図に示すように、新しくオブジェクトTake、Put、Cleanを追加する。

– Takeオブジェクトでは Using Filter と Ammountを再利用して追加する。

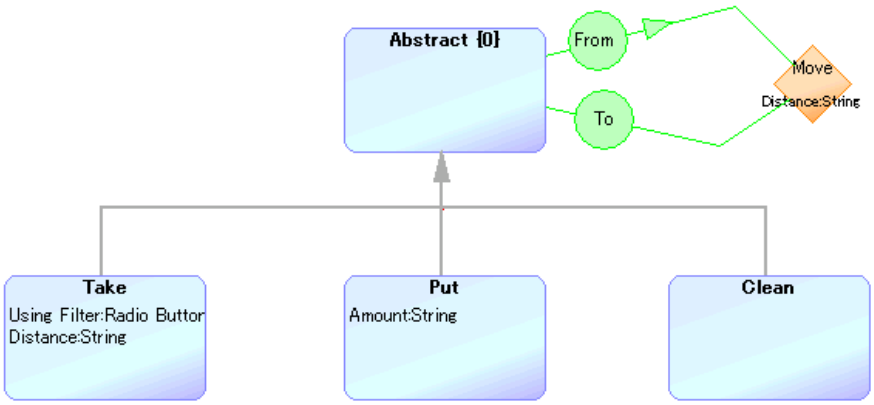
– Putオブジェクトでは Amount を再利用して追加する。

– オブジェクトAbstractを追加し、Take、Put、Cleanの原型とする。

■ 新しいリレーションシップ“Move”を追加し、Distance を再利用

■ From と To は再利用

下記メタモデルが作成できる。



・さらにルールを追加

■ Startオブジェクトを追加。Occurrence（発生回数） を 1 （最初の動作時のみ）に。

■ さらにドメインを分析：

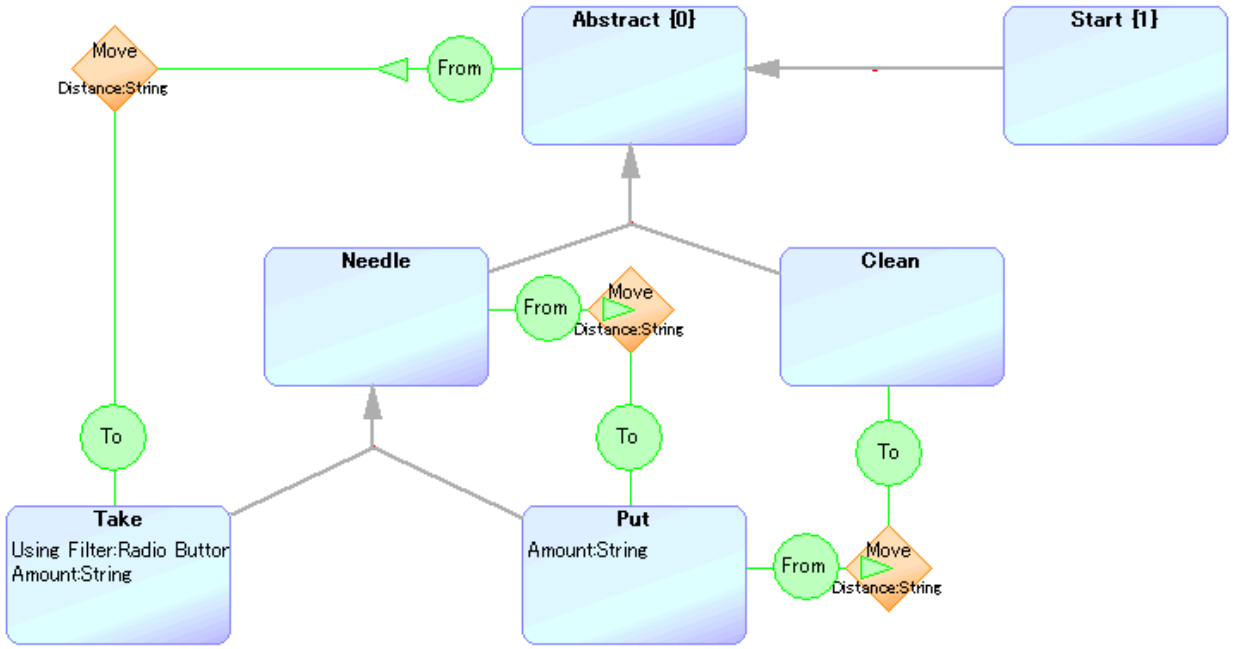
– Takeオブジェクトは、どの Abstract にも付随できる

– PutオブジェクトはTakeまたはPut に付随できる

・TakeとPutに対して、新たに上位オブジェクトのNeedleを追加

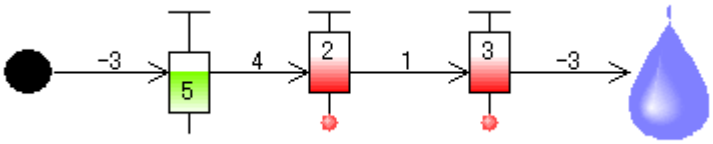
– Clean オブジェクトは、Put にのみ付随できる

各オブジェクトのプロパティは、下図のように設定します。

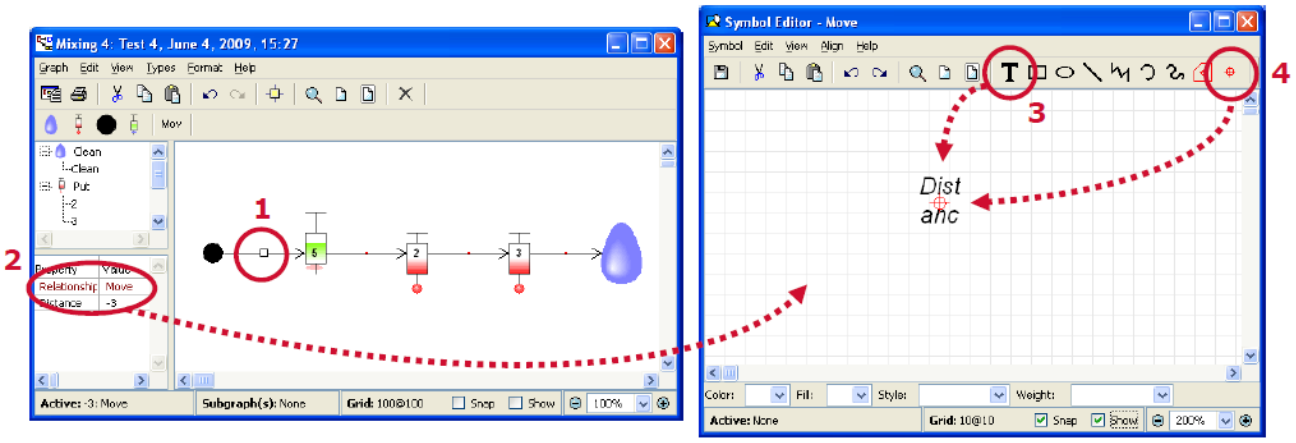


このグラフィカルなメタモデルから、新しいメタモデルを生成するために、Build を押す。

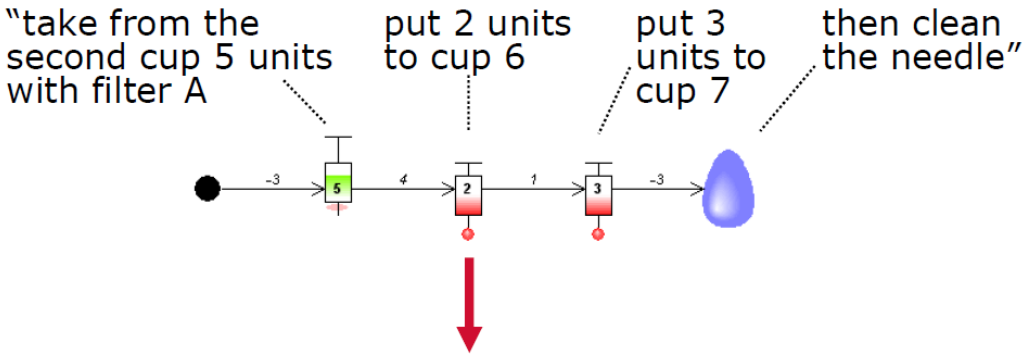
そして、下記のようなアプリケーションモデルを作成するために、ツールバーで、Create Graph ボタンをクリックし、Create Graph ダイアログのリストから“Mixing 4”を選択し、OK する（リレーションシップの数字は、この後の作業で表示できるようにします)。
下記右端にあるシンボル（Clean）に関しては、“Clean.svg”を使用しましょう。
左端にあるシンボル(Start) は黒塗りの丸を描きます。



・リレーションシップに対して、シンボルを設定
1．先程作成したダイアグラムエディタで、リレーションシップを選択
2．シンボルエディタを開く(Property シートの Relationship で shift キーを押しながらダブルクリック)
3．描画エリアで Distance プロパティの為にテキストボックスを置く(Property として Distance を設定)
4．ツールバーから Point Connectable を選択し、テキストボックスの中央でクリックする。
5．Save ボタンを押して、シンボルエディタを閉じる（リレーションシップに Distance の値が表示)



・要件 ⇒ モデル ⇒ コードは、以下のように表されます。




```
01 move(-3); filt(1); suck(5);
02 move(4); filt(0); blow(2);
03 move(1); blow(3);
04 move(-3); suck(30);
05 move(1); blow(30);
```

・ジェネレーターの設定

■ ジェネレータエディタで、ジェネレーターを実装する

– Mixing 4 のダイアグラムエディタで、メニュー“Graph->Edit Generators”を選択し、ジェネレータエディタを開く。

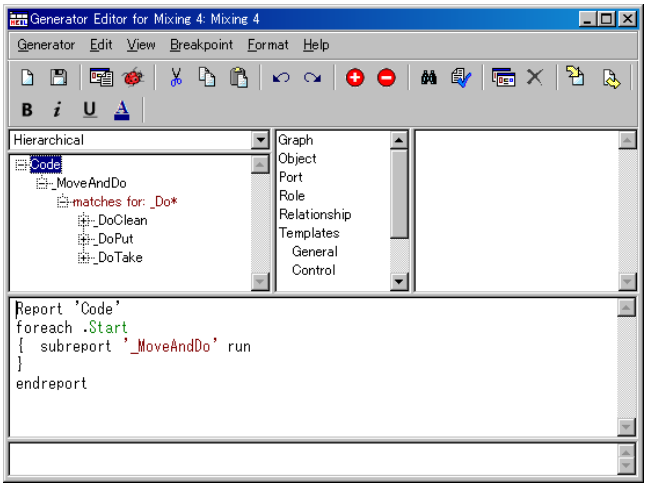
– メニュー“Generator->New”を選択する。
新しいジェネレーターの名前として“Code”を入力する。すると、編集エリアに新しい空のジェネレータースクリプトが表れます。
下記のように記述します。（ここでは、Code から呼ばれる _MoveAndDo も作成する）

```
Report 'Code'
foreach .Start                    // Start オブジェクトから開始し、
{  subreport '_MoveAndDo' run     //  '_MoveAndDo' サブジェネレーターを実行
```

```
}                                          //
endreport


Report '_MoveAndDo'
do ~From>()                                // リレーションシップに向かって、From ロールを探索
{  'move(' :Distance ');' newline          // プロパティ Distance の値を出力し、改行
                                           // “例: move(-3);”
   do ~To.()                               // オブジェクトに向かって、TO ロールを探索し、
   {    subreport '_Do' type run           // 名前が、‘_Do’＋オブジェクトタイプ の
   }                                       // サブジェネレーターを実行（例:‘_DoTake’）
}
endreport
```

■ 下記のように、他のジェネレーターも追加します。（_DoTake、_DoPut、_DoClean）



＜_DoTake：Take オブジェクトに対するジェネレーター＞

```
Report '_DoTake'
if :Using Filter; <> 'None' then   // Filter が使用されていれば、
   'filt('                         // モデルで与えられた filter 値を
   to 'AB12' translate             // コードとして適切な値に変換（A⇒1、B⇒2）し、
     :Using Filter;                //  filter コードを出力
   endto                           // “例：filt(1);”を出力し改行
   ');' newline                    //
endif
'suck(' :Amount ');' newline       // Amount 値を出力し、改行（例：suck(5);）
if :Using Filter; <> 'None' then   // Filter が使用されていれば、
   'filt(0);' newline              //  “filt(0);”を出力し、改行
```

```
endif                                   //
subreport '_MoveAndDo' run              //   '_MoveAndDo'サブジェネレーターを実行
endreport
```

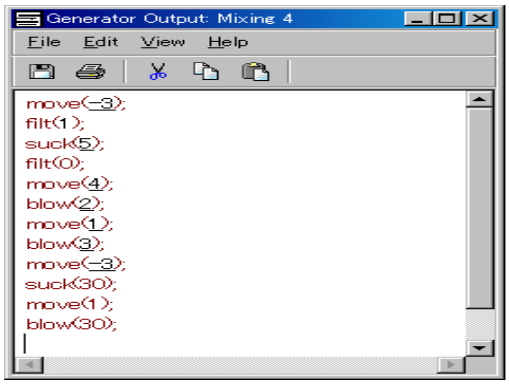
＜_DoPut：Put オブジェクトに対するジェネレーター＞

```
Report '_DoPut'
'blow(' :Amount ');' newline        // Amount 値を出力し、改行（例：blow(2);）
subreport '_MoveAndDo' run          //  '_MoveAndDo'サブジェネレーターを実行
endreport
```

＜_DoClean：Clean オブジェクトに対するジェネレーター＞

```
Report '_DoClean'                    // 注射針の洗浄に相当する固定のコードを出力
'suck(30);                           // suck(30); を出力し改行
move(1);                             //  move(1); を出力し改行
blow(30);' newline                   //  blow(30); を出力し改行
subreport '_MoveAndDo' run           //   '_MoveAndDo'サブジェネレーターを実行
endreport
```

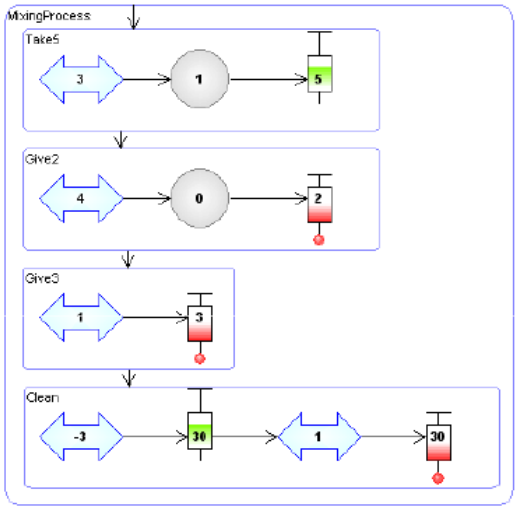
そして、メニュー“Generator->Generate”を押して、実行すると下記コードが生成されます。

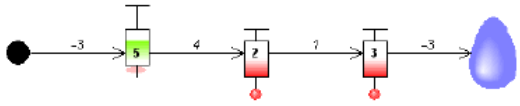


■モデリングの比較

Version4 では、モデルの抽象度を上げることで（左から右へ）、より少ない部品（Object、Relationship、Property）で、同じシステムをモデリングし、コードを生成できることがわかりました。

例えば、Clean の処理など、ただ一つのアスペクトを指定するのみになりました。





- 17 objects
- 12 relationships
- 17 properties
- **46 elements in total**

- 5 objects
- 4 relationship
- 7 properties
- **16 elements in total**

＜まとめ　以下の項目を、演習を通じて確認＞

■いくつかのモデリング言語を作成し、各バージョンで抽象度を上げた

■モデリング言語をイテレーティブに進化させながら開発できる

- メタモデルは、アプリケーションモデリングを介して検証される
- ジェネレーターは、参照となる実装に対してテストされる

■イテレーティブなアプローチにより、モデリング言語のデザインが容易

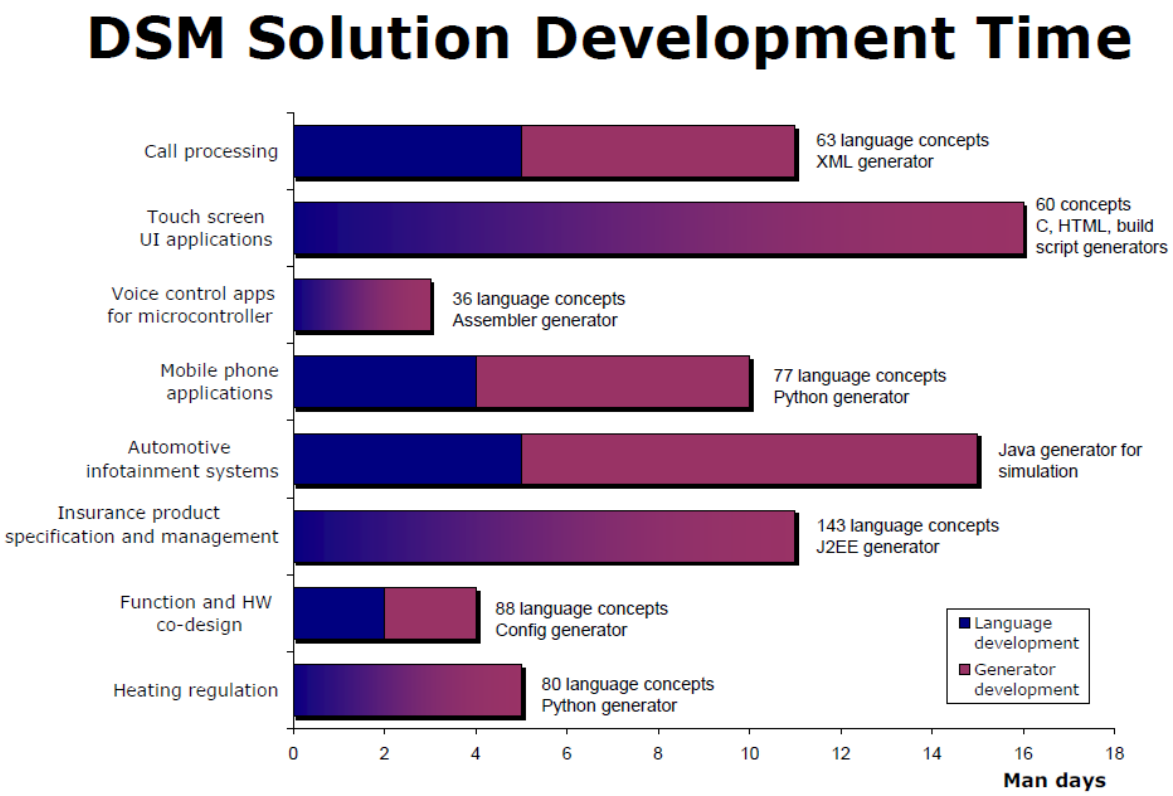
- モデリング言語に対する改善を少しずつ（リスクを最小限に）
- ドメイン熟練者が参画できる
- 言語を進化させて、新たな要求・需要に応えることができる

■メタモデリングツールにより、DSMが実現可能であることがわかる

- 道具（ツーリング）を作ることではなく、言語設計にフォーカスできる

■言語の作成に、多くの時間は掛からない

一般にはモデリング言語とコード生成機能を、ツールを用いて定義するのには多くの時間と労力が掛かると認識されています。そして大抵の場合、数年もの工数が掛かるプロジェクトになると予想されます。しかしながら我々の経験では、それは全く異なっており、ドメインによってはそれこそ数日で DSL・DSM とコード生成機能を実装しています。



すばらしい結果ですが、特別ではありません。これら事例では、MetaEdit+ を用いてモデリング言語とコード生成機能の両方を設定しています。一般に想像されるより、はるかに短期間で実装できている理由は何か？：

· 予想外となるとすれば、それは単に経験が無く、また恐らくは言語やコード生成機能を定義する、良い専用ツールを持っていなかったから。

· モデリング言語とそのコード生成機能のプロブレムドメインの範囲を限定すること。広範なドメイン、あるいは抽象度のレベルが低い場合、それは結果的に汎用的な言語を作ることになってしまい、それゆえに多くの時間が掛かることになってしまう。そして何を開発するのか明確でない場合は、自動化は厄介になるでしょう。

· 間違ったツールにより、甚大なリソースが言語やコード生成機能の開発とサポートに費やされることとなります。OOP カンファレンスで聞いた悲惨な話で、Eclipse EMF を用いてモデル言語の構築に 25 人・年の工数が費やされたとの報告がありました。そのような投資を出来る企業がそうそう有るとは思えません。ツールの違いに関して興味がありましたら、OOPSLA のワークショップ（best practices for MDSD

(Model-Driven Software Development)）での以下の資料（How much effort goes into DSL implementation?）をお勧めします。

http://www.metacase.com/blogs/jpt/blogView?showComments=true&entry=3416224823

Panasonic

設計、テスト、修正を含めて 6 時間でドメイン固有の設定を行うことができた – パナソニック電工株式会社 Laurent Safa 氏

DENSO

MetaEdit+ を用いることで、ソフトウエア開発の外部委託を削減できるようになった - 株式会社デンソー　岩井 明史氏

AUTOSAR のバージョンが変更されても、それに対する修正が簡単に行えるようになった。また、実装とテストの作業が短縮された　- 株式会社デンソー　佐藤 洋介氏

ICT

SIP ベースのサービスアプリケーションをデモするためのモデリング言語は、半日で用意できた。このスピード、MetaEdit+の使い勝手の良さは、通信オペレータ、装置ベンダに、モデル駆動でサービスを仕立てることを提案する上でのキーとなった - Tony Sloos, CTO, ICT Solutions

[参考文献とリンク]

- DSM Forum, www.dsmforum.org
- Blogs: www.metacase.com/blogs
- Brinkkemper, S., Lyytinen, K., Welke, R., Method Engineering -Principles of method construction and tool support, Chapman & Hall, 1996
- Kelly, S., Tolvanen, J.-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley, 2008. http://dsmbook.com
- Kieburtz, R. et al., A Software Engineering Experiment in Software Component Generation, Proceedings of 18th International Conference on Software Engineering, Berlin, IEEE Computer Society Press, 1996.
- Sprinkle et al (eds) IEEE Software, DSL&M special issue, July/Aug, 2009
- Tolvanen, J.-P., Kelly, S., Defining Domain-Specific Modelling Languages to Automate Product Derivation: Collected Experiences. Procs of the 9th International Software Product Line Conference, Springer-Verlag, 2005.
- Weiss, D., Lai, C. T. R., Software Product-line Engineering, Addison Wesley Longman, 1999.