



MetaEdit+

専用モデル環境の作り方



この資料は、MetaEdit+ に標準でインストールされるDemo環境のWatch Exampleを使用して、モデル環境を構築する方法とそのモデル環境を使って作成されたモデルからソースコードを生成する方法について説明しています。モデルやメタモデルを作成する為の具体的な操作手順は、

Evaluation tutorial (**Family Tree のモデリング例**)

http://www.metacase.com/support/45/manuals/evaltut/et-Preface_.html

MetaEdit+ User's Guide

<http://www.metacase.com/support/45/manuals/meplus/Mp.htm>

1 等をご参照ください。



目的

- MetaEdit+ に標準でインストールされるDemo環境のWatch Exampleを使用して、MetaEdit+ の以下の機能を理解する
 - メタモデルの設定・構築方法
 - モデルの作成方法
 - ソースコードとドキュメントの生成方法
 - ソースコードとドキュメントを自動生成する為のGeneratorの設定(記述)方法



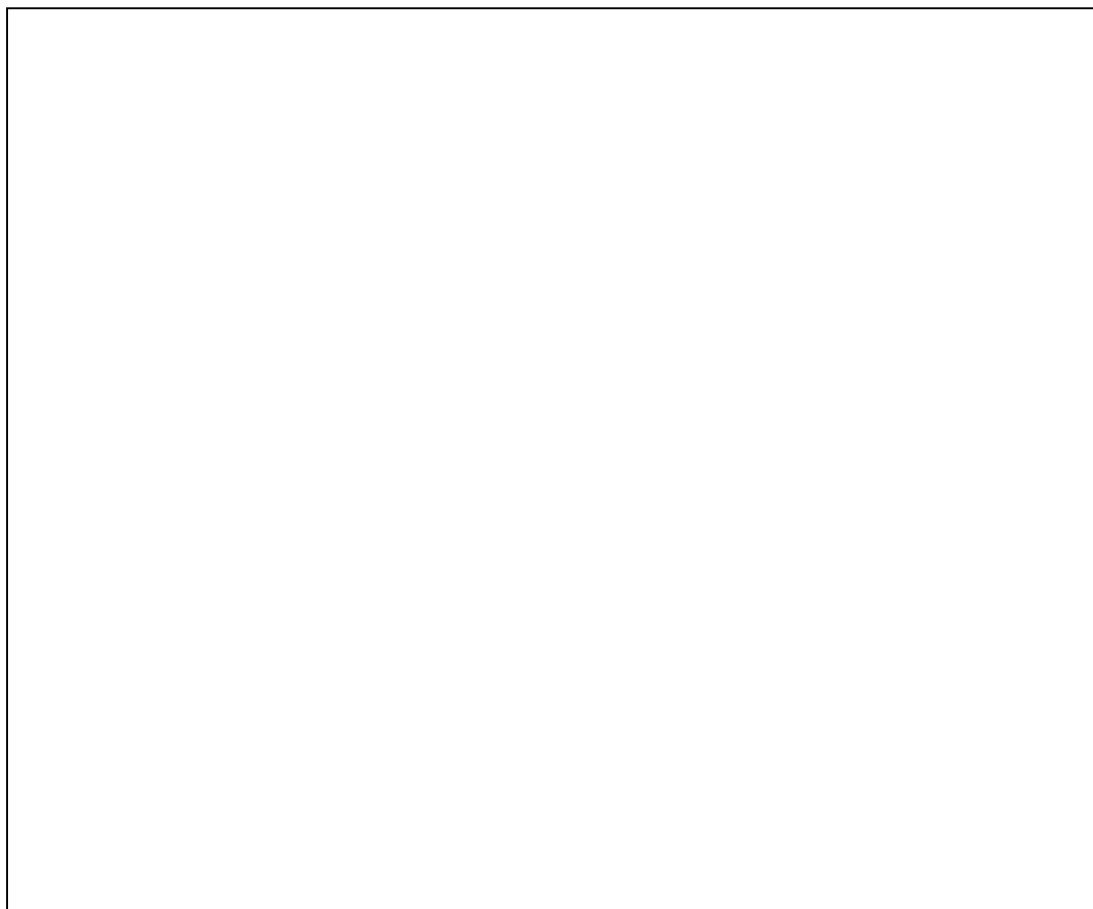
アジェンダ

- MetaEdit+の実行
- メタモデル言語について
- ソースコード(ドキュメント)の生成
- モデルの修正
- モデリング環境の構築
 - グラフィカルなメタモデリング環境
 - メタモデルのインポートとエクスポート
- モデルの作成
- ソースコード生成環境の構築





MetaEdit+の実行





レポジトリの選択

- MetaEdit+は、全てのデータをレポジトリとして管理しており、レポジトリを選択して、ログインする事からすべての作業が始る
- MetaEdit+ 4.5
を実行



© 2007 MetaCase

5

MetaEdit+は全てのデータをレポジトリとして、マルチユーザー対応のデータベース管理をしています。従って、データにアクセスする為には、データベースにログインする(使用するユーザーを特定する)必要があります。MetaEdit+の全てのツールは、データベースにログインしない限り使えません。

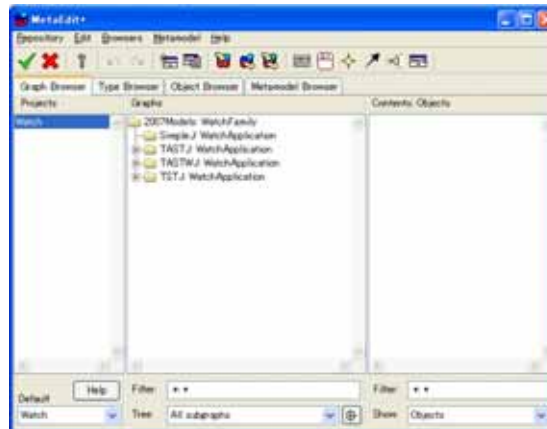
これは、ユーザーの権限を制限する為に必要な処理であり、管理者(メタモデラー)以外のユーザーが不用意にデータを削除することを防いでいます。



ログイン

WatchExampleで作業

- Log in to Repository: demo
Open Projects: Watch
Repository User: Sysadminを選択しLoginする



© 2007 MetaCase

6

“Log in to Repository”を“demo”、“Open Projects”を“Watch”、
”Repoisitory User”を”Sysadmin”に設定しLoginボタンを押します。
メインランチャーが開き、WatchExampleが読み込まれます。このメインランチャーがMetaEdit+の操作の全ての基本になります。



ソースコード(ドキュメント)の生成

今回使用するサンプルでは、モデルからのソースコードの自動生成に加えて、生成したコードをコンパイル、実行するスクリプトまで自動生成します。その為、先ずコンパイル環境が正しく構築されていることを確認する為に、通常のサンプルを使って、ソースコードを自動生成することから始めます。このコンパイル環境は、後ほど紹介するモデリング環境の構築からモデル作成、コード生成の説明でも使用します。

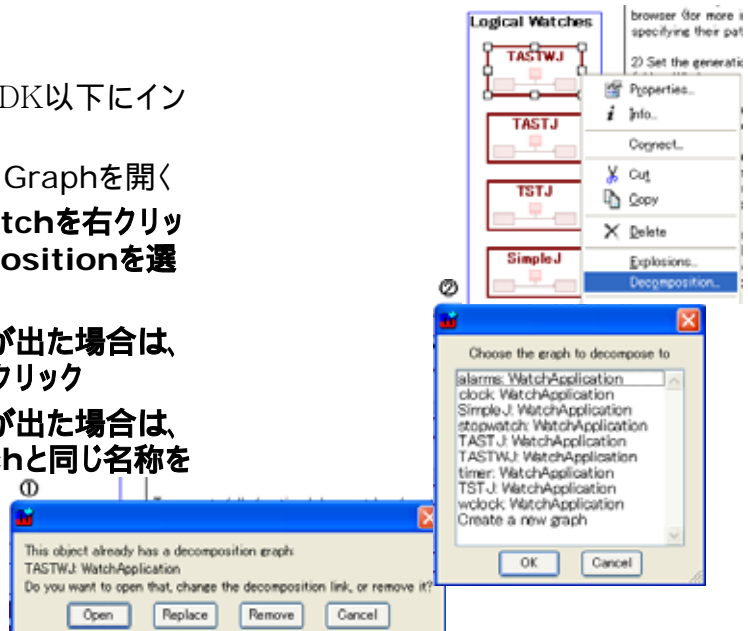
コード生成は後回しにしたい方は、スライド10へ進んで下さい。



ソースコード自動生成 – モデルの確認

WatchExampleで作業

- JAVA SDKを
C:\¥JAVA¥J2SDK以下にインストール
- 2007Models Graphを開く
- 各LogicalWatchを右クリックし、Decompositionを選択
- のメッセージが出た場合は、Openボタンをクリック
- のメッセージが出た場合は、LogicalWatchと同じ名称を選択



© 2007 MetaCase

8

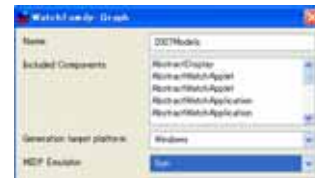
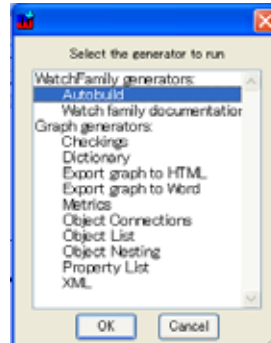
このサンプルのソースコードはJAVA言語で記述されています。その為、JAVAの開発環境をインストールする必要があります。最新のものをインストールしてください。又、生成したソースコードからコンパイラを呼び出すため、JAVAのコンパイラが特定のディレクトリにインストールされている必要があります。もし、既に環境がインストールされている場合は、C:\¥JAVA¥J2SDKディレクトリ以下にコピーを作成してください。開発環境がインストール出来れば、ソースコード自動生成の為に、モデル環境を確認します。



ソースコード自動生成 — 自動生成と実行

WatchExampleで作業

- Graph->Propertyを選択
- Generation Target PlatformがWindows、MIDP EmulatorがSunになっていることを確認
- Graph->Generateを選択
- Autobuildを選択してOKボタンをクリック



Graph->Propertyでの設定は、このサンプルのモデリング環境特有のもので、マルチプラットフォームやマルチ言語環境に対して、ソースコードを生成させる為の具体的な例になります。トップレベルに設けた、システム全体を定義する為のグラフに対して、プラットフォームや言語を選択する為のプロパティを設けています。Graph->Generateで作成されているジェネレータの一覧を表示させ、ジェネレータを選択して実行します。



モデリング環境の構築

メタモデル言語について

先ず、MetaEdit+によるモデリングを行う為には、モデリング環境の構築が必要になります。ここでは、モデリング環境を構成するメタモデル言語について解説します。



GOPPRR はメタモデルを作るための言語

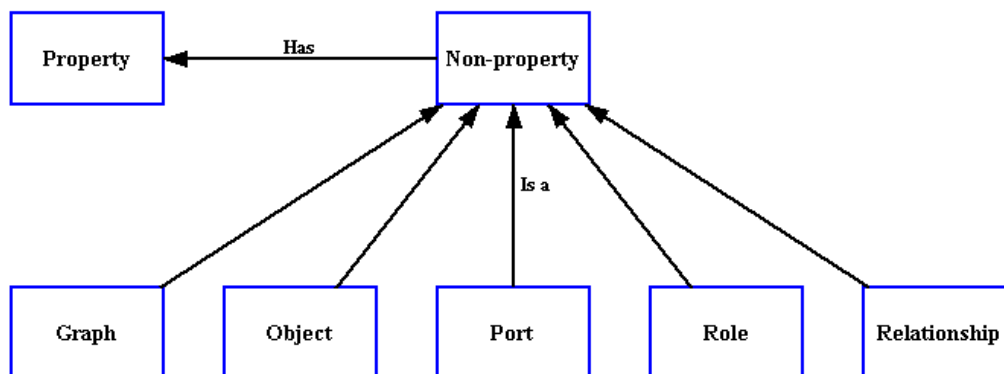
Graph, Object, Property, Port, Role, Relationship

- MetaEdit+ Workbenchを用いてモデリング言語のGOPPRR を設定し、専用モデル環境を構築する
- **3層構造:各々が一つ前の層のインスタンスになっている**
 - GOPPRR → **メタタイプ** → インスタンス
 - *MetaEdit+* → *WatchApplication* → *Stopwatch*
- GOPPRRでは様々なモデルエレメント間のリンクも定義する

MetaEdit+ では、GOPPRRという言語使用して、モデリング環境を構築します。言語は、グラフ、オブジェクト、プロパティ、ポート、ロール、リレーションシップで構成されています。



Graph, Object, Port, Role, Relationship 全てが Property を持つことが出来る

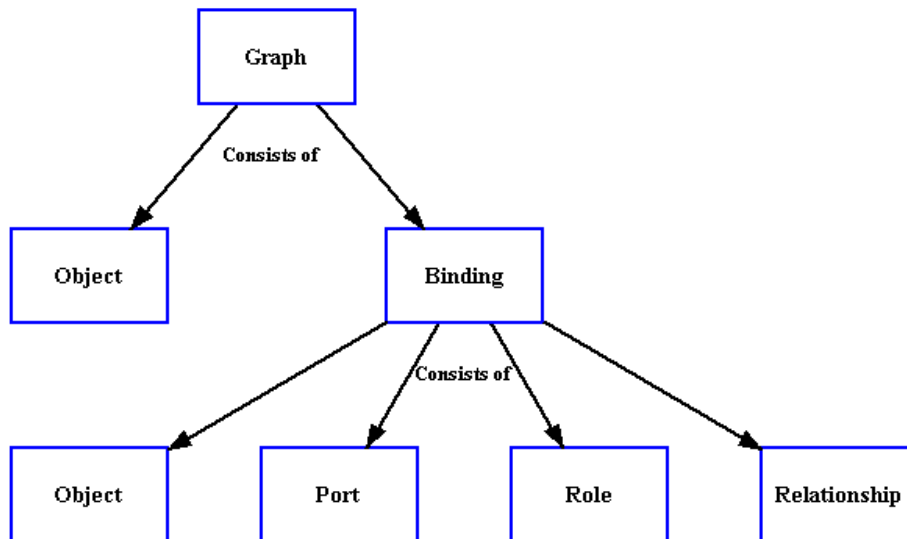


これらの property は、MetaEdit+ のGUIを介して、新規追加、修正、表示が容易に出来る。property の数、複雑さ、階層の深さなどの制約は無し。オブジェクト指向のクラスタイプなどのproperty にも対応。

プロパティ以外の全ての要素は、プロパティを持つ事が出来ます。プロパティが他の要素から独立して存在しており、各要素に自由にリンクできる事が、MetaEdit+ が少ない要素でメタモデルを構成できる理由です。



Graph は、Object と Binding で構成される

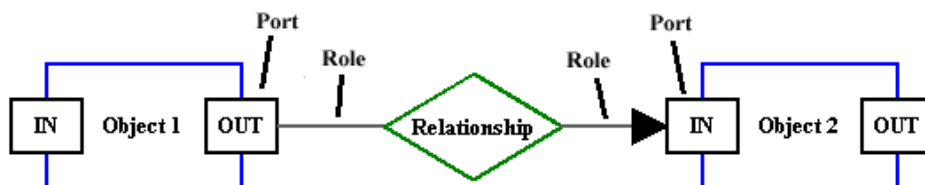


メタモデルの構造を表しています。各要素の詳細は、以降のスライドで説明します。



Binding の内訳

graph 内で、objects, ports, roles, relationships が以下のように接続される



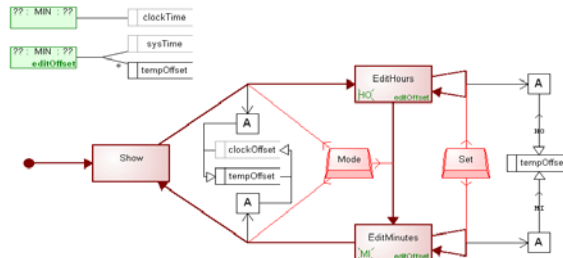
各 binding は、1つの relationship、2つ以上の role、role ごとに1つの object (さらに随意的に1つの port) で構成される。大抵は port は使用されることなく、role は直接 object へ接続される

バインドイングの構成要因を示しています。



Graph

- Graphはobjectとそのbindingで構成される
- bindingはobject、port、role及びrelationshipの結びつきを規定する
- object、port、role及びrelationshipは他のgraphで再利用できる
- Graphは1つのモデリング技法になる



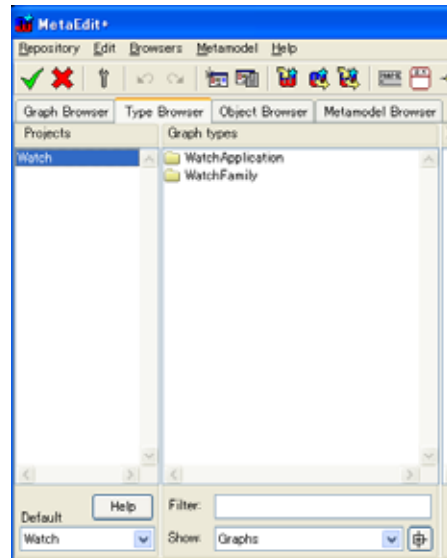
全てのメタモデルはグラフ上に配置されます。また、配置されることによって、モデリング環境内に、各メタモデルのインスタンスが生成されます。

インスタンス化されたメタモデルは、他のグラフで再利用する事が出来ます。



GraphType

- Graphには必ず1つの GraphTypeが結びついている
 - GraphTypeが、ObjectとBindingを規定している
 - Graphに対する Propertyも GraphTypeに対して規定している
- Propertyとして、プラットフォームや生成言語の種類を設定している例もある

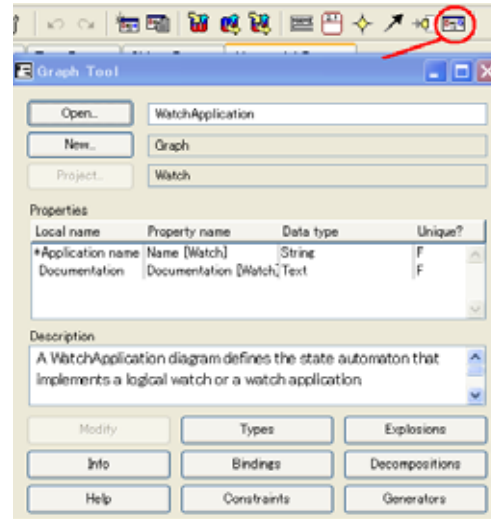


オブジェクトやバインディング、グラフに対するプロパティと言った、メタモデルはグラフタイプに対して規定されています。グラフ作成時にグラフタイプを指定する事で、そのグラフへのモデリング環境の設定がなされます。そして、グラフ上へのモデルの記述が可能になります。



GraphTypeの設定と作成

- GraphToolボタンで GraphTypeの設定が可能
- Openの横のテキストボックスに、未定義の名前を入力することで、新規に作成が可能
- Typesボタンでメタモデルの一覧を表示
- Bindingsボタンで、設定されているバインディングを表示

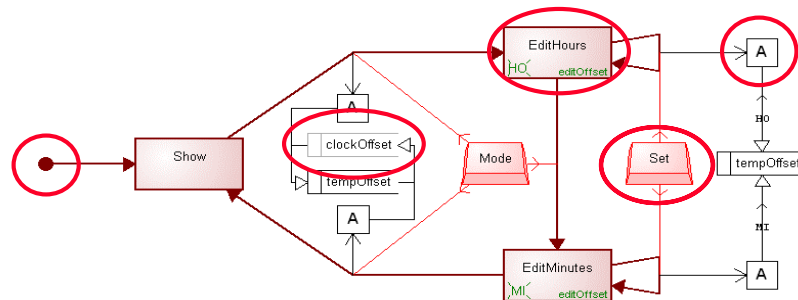


具体的なグラフタイプの作成と設定の手順です。グラフツールを使って作成、設定します。OPENボタンの横のテキストボックスが、グラフタイプの名前です。OPENボタンで既存のグラフタイプが選択でき、編集が可能になります。テキストボックスに未定義の名前を入力する事で、新規にグラフタイプを作成出来ます。Typesボタンでメタモデルの一覧を表示し、表示画面から各メタモデルの追加、削除、修正が出来ます。Bindingsボタンで設定されているバインディングの一覧を表示し、表示画面から構成要素の追加、削除、修正が出来ます。



Object

- relationshipやroleと無関係に、それ自身単独で配置可能なもの
- オブジェクトの特徴を示すためにPropertyを設定可能
- 例
 - WatchApplication diagram: State, button, display function

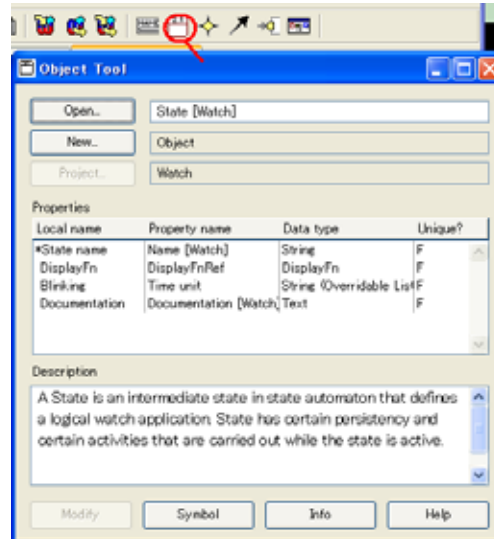


それ自身が単独で存在しうるものをオブジェクトで表現します。オブジェクトのインスタンスを生成し、名前を付ける事でオブジェクトの特徴を表します。オブジェクトの特徴を更に詳細に表現する為にプロパティーを設定することも出来ます。各オブジェクトのインスタンスを区別する為に、シンボルを設定することも可能です。具体的には、図中のステート、ボタン、アクション、変数等がこれにあたります一般的なアプリケーションにおけるダイアログ、表示画面等を表現することもあります。



Objectの設定

- Metamodel BrowserタブでShowをObjectTypeに設定することで確認可能
- 表示されるObjectをダブルクリックすることで、Objectの詳細設定が可能
- ObjectToolボタンのクリックでも表示可能
- Open横のテキストボックスに未定義の名前を入力することで新規作成可能

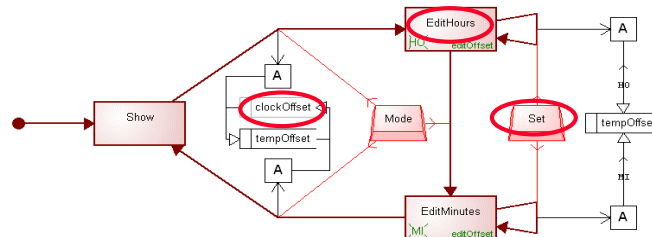


具体的なオブジェクトの作成と設定の手順です。オブジェクトツールを使って作成、設定します。OPENボタンの横のテキストボックスが、オブジェクトの名前です。OPENボタンで既存のオブジェクトが選択でき、編集が可能になります。テキストボックスに未定義の名前を入力する事で、新規にオブジェクトを作成出来ます。Propertiesリストに、オブジェクトの特徴を示す為のプロパティーを設定出来ます。Symbolボタンでシンボルを作成、編集出来ます。



Property

- PropertiesはProperty以外 (graph, object , port , relationship及びrole)のインスタンスの特徴を示す
 - モデルにデータを結び付ける為の最も重要なコンセプト
 - Project内に独立して存在する
- 例
 - WatchApplication: state , button , timeunit等の名前
 - UML Class diagram: Name , stereotype , constraints

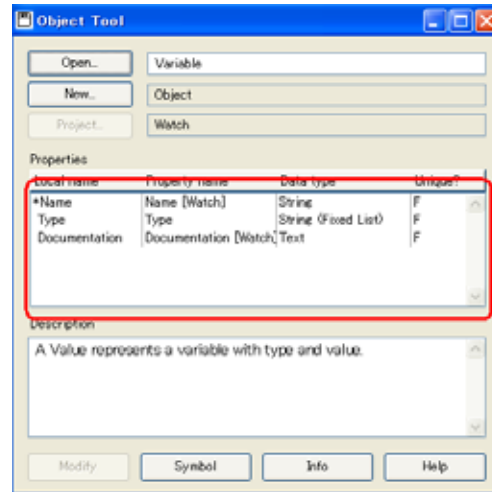


プロパティ以外のメタモデル(グラフ、オブジェクト、ポート、リレーションシップ、モデル)の特徴を示す為に用いられます。モデルにデータを結びつけたり、モデルを具体化する為に重要な部分です。プロパティはメタモデル上で直接定義されるわけではなく、独立して存在します。各メタモデルは、プロパティをリンクする事で結び付けます。



Propertyの結びつけ

- ObjectTool, RelationshipTool、RoleTool, GraphTool使って、Propertiesリストボックスにセットする事で結び付けられる
- リストボックス右クリックで、Add Propertyで選択可能
- 選択時にNew Property Typeを選択する事で新規作成可能
- ここで、ID、Uniquenessが設定可能

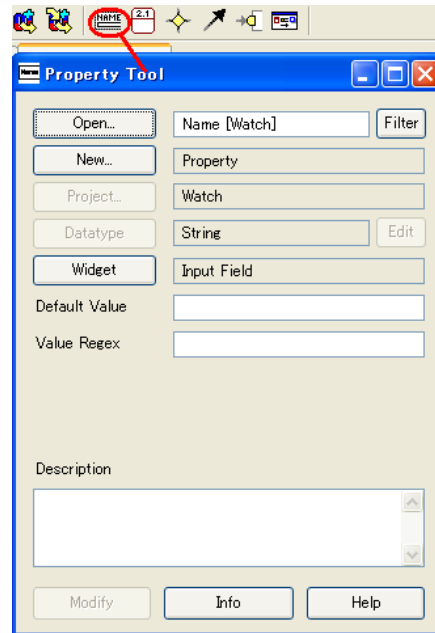


プロパティは各メタモデルツールで結びつける事が出来ます。既存のものを結び付ける代わりに、ここで新規にプロパティを作成することも出来ます。



Property設定

- PropertyToolで各値を設定可能
- **DataType**、**Widget**、**DefaultValue**、**ValueRegex**、**LocalName**が設定可能
- **Open**横のテキストボックスに未定義の名称を設定することで新規に作成可能



プロパティとして設定できる具体的な項目です。DataType 以下の中から選択可能 String、text、number、boolean、Graph、Object、Port、Relationship、Role、collection(集合) Widget 値を入力するためのUI Value Regex 入力された値の文字としての有効条件正規表現で設定可能 LocalName 使用しているプロパティにオブジェクト内の別名を設定可能同じプロパティを、同一オブジェクト内で複数使用可能にする



Port

- Portはobjectの特徴を示すもので、objectに接続するroleの意味を規定する
- Portはpropertyを持つことができる
- 例: objectのINとOUTのport

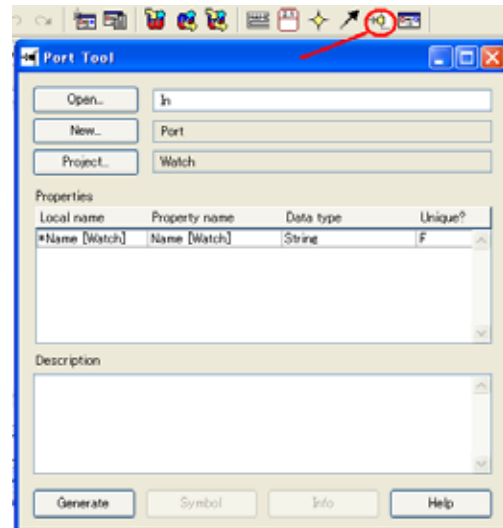


ポートはオブジェクトにロールを接続する際に、その意味を規定する為に使用します。



Portの設定と新規作成

- Port Toolで設定、作成が可能
- Open横テキストボックスに未定義の名称を入力することで新規作成が可能
- シンボルを設定可能



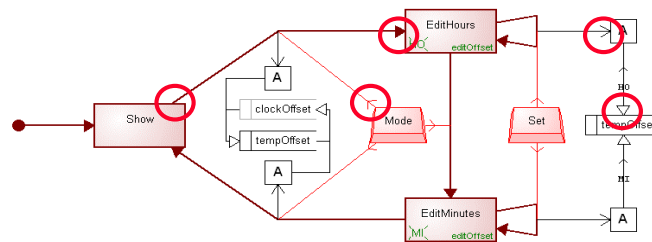
具体的なポートの作成と設定の手順です。ポートツールを使って作成、設定します。OPENボタンの横のテキストボックスが、ポートの名前です。OPENボタンで既存のポートが選択でき、編集が可能になります。テキストボックスに未定義の名称を入力する事で、新規にポートを作成出来ます。

Propertiesリストに、ポートの特徴を示す為のプロパティを設定できます。Symbolボタンでシンボルを作成、編集出来ます。



Role

- objectがそのrelationshipにどの様に関与するかを規定する
 - objectの端に配置され、線とシンボルで表示される
- 例
 - WatchApplication diagram: From , To , Action-Body, Set
 - UML Class diagram: Super-class, Sub-class

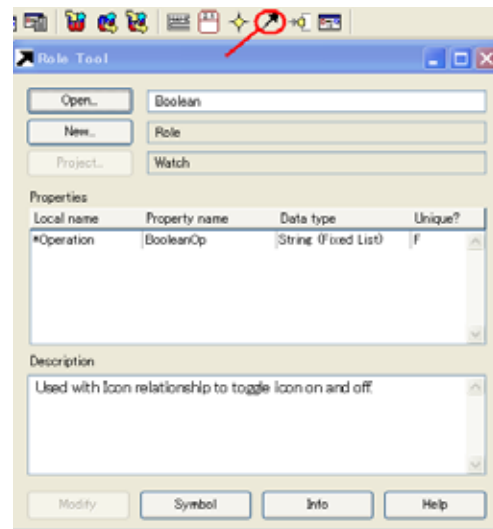


バインディングにおけるオブジェクトの役割を示します。



Roleの設定

- Role Toolで設定、作成が可能
- Open横テキストボックスに未定義の名称を入力することで新規作成が可能
- シンボルを設定可能

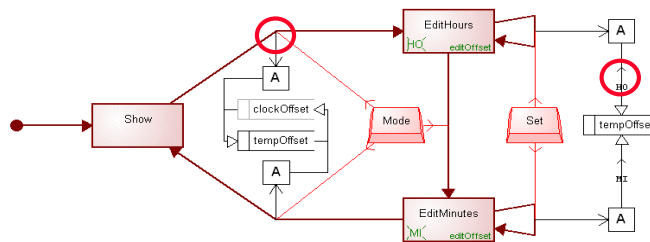


具体的なロールの作成と設定の手順です。ロールツールを使って作成、設定します。OPENボタンの横のテキストボックスが、ロールの名前です。OPENボタンで既存のロールが選択でき、編集が可能になります。テキストボックスに未定義の名前を入力する事で、新規にロールを作成出来ます。Propertiesリストに、ロールの特徴を示す為のプロパティを設定できます。Symbolボタンでシンボルを作成、編集出来ます。



Relationship

- Relationshipは、2つ以上のobjectを、意味を持たせて接続する
 - Roleの中央の点で表示され、シンボルを持つことも出来る
- 例
 - WatchApplication diagram: State transition, Roll
 - UML Class diagram: Inheritance, Association

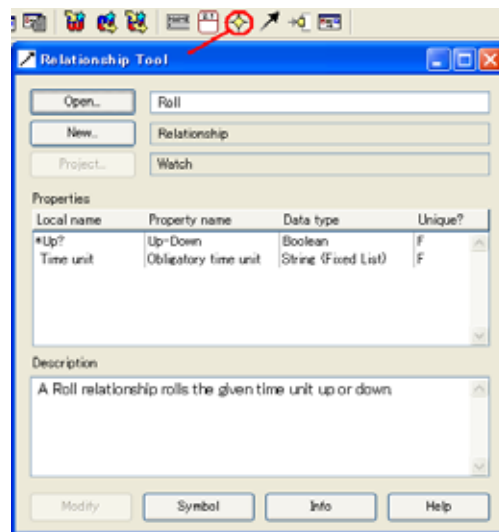


2つ以上のオブジェクトを接続する事でその関係を表します。ロールのオブジェクトと逆側の接続点であり、複数のロールが結合されます。



Relationshipの設定

- Relationship Toolで設定、作成が可能
- Open横テキストボックスに未定義の名称を入力することで新規作成が可能
- シンボルを設定可能



具体的なリレーションシップの作成と設定の手順です。リレーションシップツールを使って作成、設定します。OPENボタンの横のテキストボックスが、リレーションシップの名前です。OPENボタンで既存のリレーションシップが選択でき、編集が可能になります。テキストボックスに未定義の名前を入力する事で、新規にリレーションシップを作成出来ます。Propertiesリストに、リレーションシップの特徴を示す為のプロパティを設定できます。Symbolボタンでシンボルを作成、編集出来ます。



モデリング環境の構築

グラフィカルなメタモデリング環境

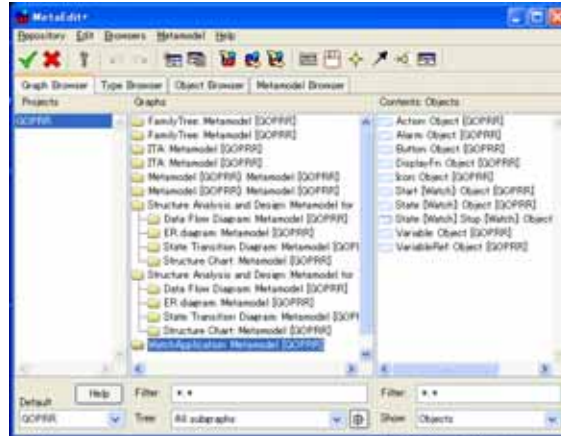
MetaEdit+ には、ここまで説明したメタモデルの設定をグラフィカルに行う為のツールも用意しています。



GOPRRの呼び出し

WatchExampleで作業

- RepositoryからGOPRRを選択し、ログインする
- WatchApplicationを開く

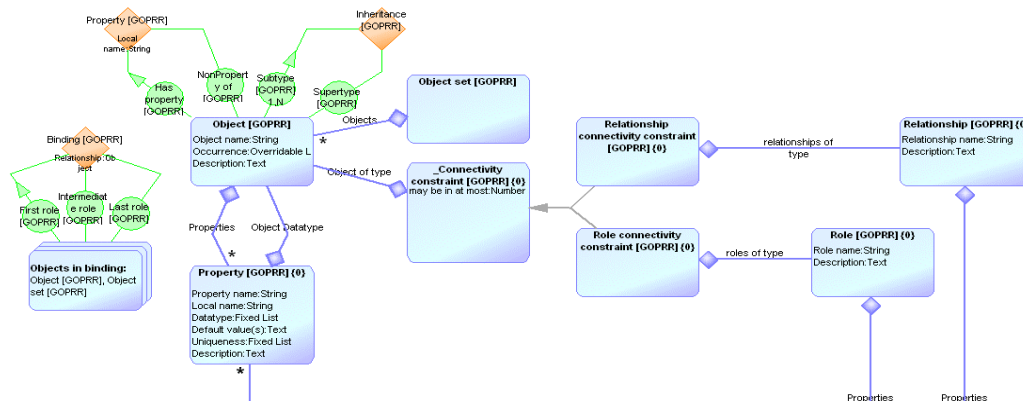


MetaEdit+ を実行し、GOPRRにログインする事で、グラフィカルメタモデリングが行えます。メインランチャーのGraphsにあるWatchApplicationが、WatchExampleのグラフィカルメタモデルです。



グラフィカルGOPRRメタモデル言語

- 図形で表現可能なGOPRR
 - 現状ではPortは未サポート、その為名前はGOPRR
- 例: グラフィカルGOPRR言語のメタモデル



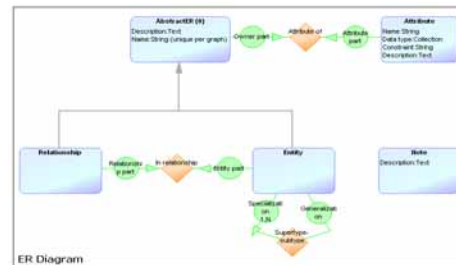
GOPRRをグラフィカルに表現する為の環境です。現状はポートをサポートしてない為、グラフィカルGOPRRメタモデル言語(以降GOPRR)と呼んでいます



Graph - GOPRR

■ Graph

- 1つのGraphTypeに対して、1つのGraphが描かれる
 - Graphのproperty
 - Objects, relationships, rolesとそのproperty
 - Binding
 - その他の制約を含む
- サブグラフへのリンクを含む
"Metamodel for multiple graphs"使えば、Graphをオブジェクトとして扱うことも可能



GOPRRに於けるグラフは、モデリング環境に於けるグラフタイプに相当します。グラフ上にオブジェクトやバインディングを設定する事でメタモデルを作成してゆきます。GOPRRでもグラフの作成にはグラフタイプを選択します。選択できるグラフタイプは Metamodel[GOPRR](メタモデルを構築する為の環境) Metamodel for multiple graphs[GOPRR](複数のグラフタイプ間の関係を表現する為の環境)の2種類です。



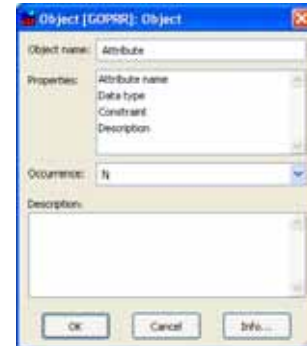
Object - GOPPR

■ Object

– 定義項目:

- 名前
- Property
- 抑制

Attribute
Name:String
Data type:Collection
Constraint:String
Description:Text



■ Object sets

- 複数Objectの集まり
- 幾つかのobjectが、同じroleで接続されるような場合のbindingを定義するとき使用される



GOPRRに於けるオブジェクトはGOPRRのオブジェクトと同じです。名前を設定する事で、メタモデルとして実体化されます。オブジェクトツールのPropertiesと同じ方法で、プロパティを設定できます。又、幾つかのオブジェクトが同じロールで接続されるような場合には、複数の定義済みオブジェクトを纏めてオブジェクトセットとして定義することも可能です。



Relationship&Role - GOPRR

■ Relationship

- 定義内容
 - 名前
 - Property
- Bindingの一部として定義される

Attribute of



■ Role

- 定義内容
 - 名前
 - Property
- Bindingの一部として定義される

Attribute part



GOPRRに於けるリレーションシップとロールもGOPRRと同じです。名前を設定する事で、メタモデルとして実体化されます。リレーションシップ・ロールツールのPropertiesと同じ方法で、プロパティを設定できます。



Property - GOPRR

- Property
 - グラフィカルな表示無しに作成される
 - 定義内容
 - 名前
 - Local name
 - データ型
 - 初期値
 - 一意性の抑制
 - graph、object、relationship**或いはrole**の一部として表される
 - Objectは、そのシンボル内にPropertyを表示する

GOPRRに於けるプロパティーもGOPRRと同じです。グラフィカルな表示は持たず、他のメタモデルに結びつける事で実体化します。



■ Property

- 他のobjectをpropertyとして使用する場合は、Property relationshipを使用する



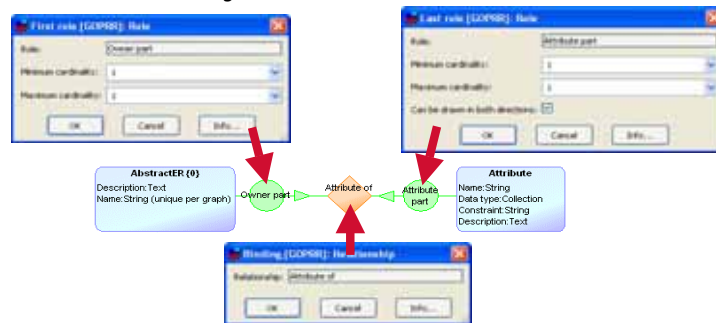
あるオブジェクトを他のオブジェクトのプロパティとして表示する場合は、プロパティリレーションシップを使って表現できます。



Binding - GOPRR

■ Binding

- object間のrelationshipとして定義される
- Binding relationshipはpropertyとしてRelationshipを持つ
- Roleはpropertyとしてroleを持つ
- RoleではCardinality(接続数)を定義できる



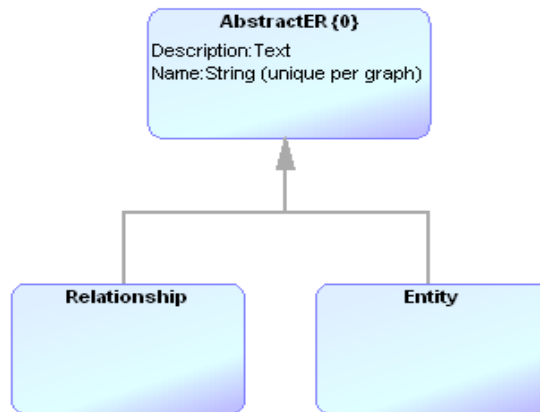
GOPRRに於いて、バインディングはBindingリレーションシップで表現されます。Bindingリレーションシップは、モデリング環境でのリレーションシップ名をプロパティとして持ちます。Bindingリレーションシップは、複数のロールを持ちますが、各ロールがモデリング環境でのロール名をプロパティとして持ちます。



Inheritance - GOPPR

■ 継承

- Objectの継承は、Inheritance relationshipを使って定義できる



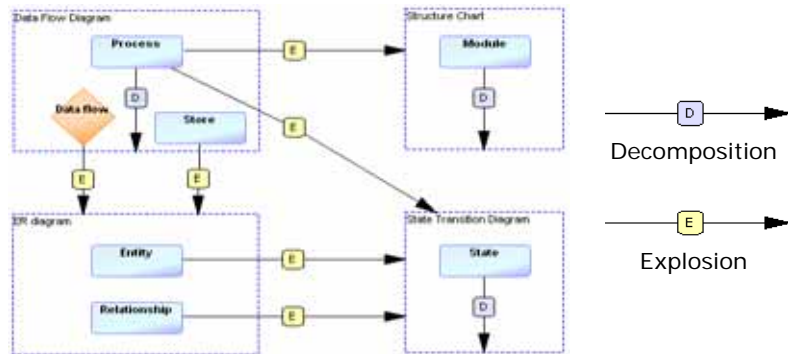
Meta Edit + ではオブジェクトに継承を持たせる事ができます。GOPPRでは、**Inheritanceリレーションシップ**を使って定義できます。



サブグラフ - GOPRR

■ サブグラフ

- Decompositionとexplosionのリンクは“Metamodel for multiple graphs”ダイアグラム使って定義できる

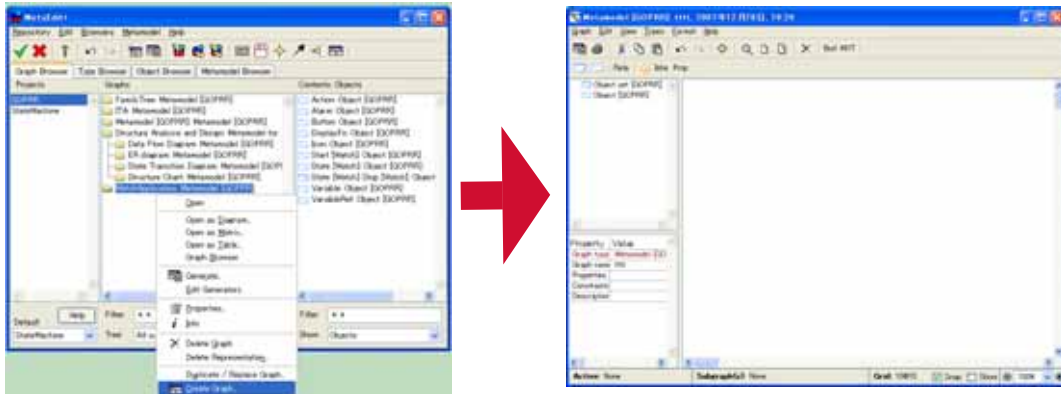


Metamodel for multiple graphsを使ったグラフでは、グラフタイプ間の関係を表す事が出来ます。各メタモデルに関連付けすることも可能です。表現できる関係は、Decomposition(オブジェクトとグラフを一对一で関連付) explosion(オブジェクト、リレーションシップ、ロールとグラフを一对多で関連付)の2種類です。



モデリング環境の作成 - GOPRR

- グラフィカルメタモデリング環境の作成
 - Graphs領域で右クリックし、CreateGraphを選択
 - CreateGraphダイアログでMetaModel[GOPRR]を選択
 - 次のダイアログに作成するグラフタイプの名前を入力(GraphName)

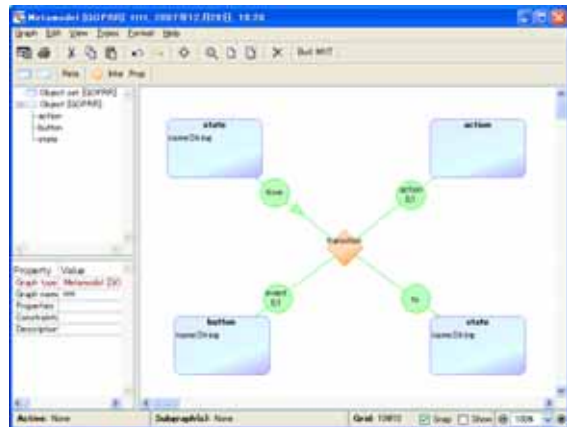


新規のグラフィカルメタモデリング環境は、Graphs領域で右クリックし、ポップアップメニューからCreateGraphを選択して作成します。CreateGraphダイアログで、作成したいメタモデリング環境を選択します。(MetaModel[GOPRR]、MetaModel for multiple graphs [GOPRR]のどちらかを選択) 次に表示されるダイアログがメタモデリング環境のグラフタイプの設定画面です。グラフタイプの名前やプロパティー等を設定します。



モデリング環境の作成 - GOPRR

- 使用するモデル部品は、オブジェクトとリレーションシップ
 - オブジェクトとオブジェクトセット
 - バインディング、インヘリタンス、プロパティーの各リレーションシップ
- これらを使って、ドメインの機能を表現する
- ボタンを押すことでステートが遷移し
その際にアクションが実行される
事を表現した例



メタモデリング画面では、オブジェクトとリレーションシップを使って、ドメインの機能を表現します。この例では、ボタンを押すことでステートが遷移し、アクションが実行される様子表現しています。



モデリング環境の構築

メタモデルのエクスポートとインポート

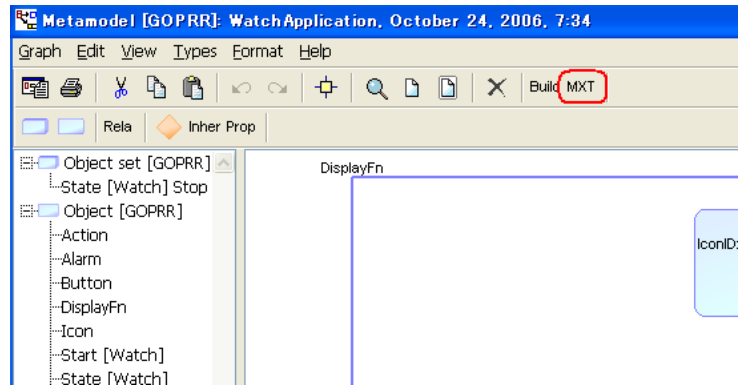
GOPRRで作成したメタモデルをモデリング環境にインポートする手順を紹介します。



MXTファイルの作成 (メタモデルのエクスポート)

WatchExampleで作業

- MXTボタンクリックでreportディレクトリに WatchApplication.mxtファイルが生成される
- GOPRRで作成したメタモデルを、通常モデリング環境にインポートすることが出来る



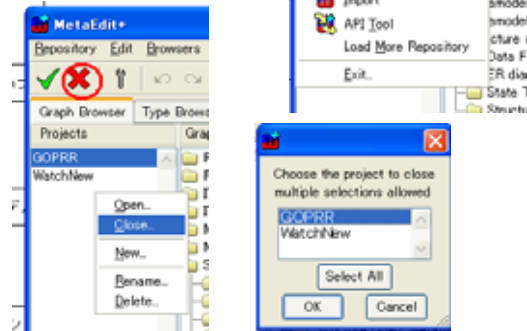
“グラフィカルなメタモデリング環境の構築“の最初で開いたWatch Applicationを使って作業を進めます。GOPRRのグラフにあるMXTボタンで、作成したメタモデルがエクスポートされます。これをモデリング環境として新規のプロジェクトにインポートして、モデリング作業を行うこととなります。



新しいProjectの作成

WatchExampleで作業

- Repository->NewProjectで WatchNewを作成
- Projectsウィンドウ右クリック ->Close
- ダイアログでGOPRRを選択してOK
- Abandonをクリック



新規のプロジェクトを作成する手順です。



Metamodelのインポート

WatchExampleで作業

- Importボタンをクリック
- WatchApplication.mxtを選択
- MetaModelとGraphTypeがインポートされる



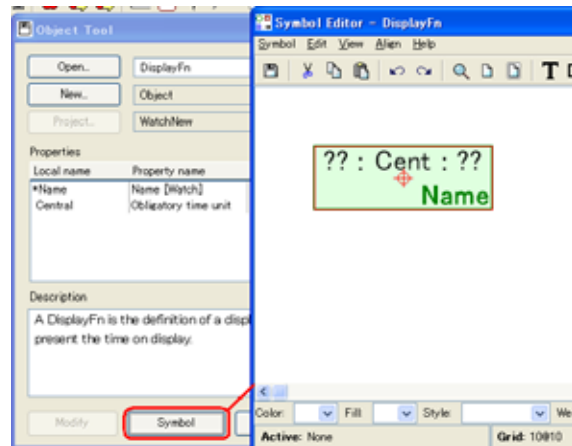
メタモデルのインポートの手順です。



シンボルの設定

WatchExampleで作業

- MXTファイルをインポートした時点では、デフォルト状態のMetaModelのみが登録されている
- このままではメタモデルの違いが分からない -> シンボルを設定
- 各メタモデルの設定画面からsymbolボタンをクリック -> シンボルを設定
- 既に設定されているシンボルをインポートすることも可能

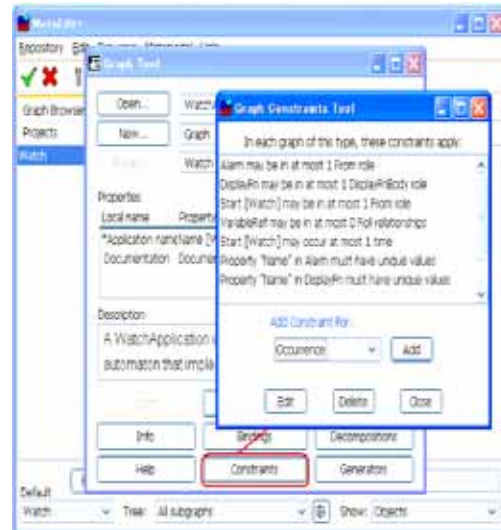


GOPRRで作成したメタモデルをインポートした場合は、オブジェクトは中央に名前の書かれた四角リレーションシップは小さな点ロールは線のみで表されています(デフォルト状態)このままでは各メタモデルが区別できない為、シンボルを設定する必要があります。各メタモデルツールからシンボルエディタを立ち上げ、シンボルをデザインできます。Symbol->ImportSVG(ExportSVG)で、シンボルのインポート・エクスポートも可能です。(既存のメタモデルからエクスポートしたシンボルを使用すると効率的です)



コンストレインツの設定

- グラフ毎のコンストレインツを設定可能
 - オブジェクトに接続されるロールやリレーションシップの数
 - グラフ内に配置できるオブジェクトの数
 - グラフ内で使用するオブジェクトのプロパティに一意性



グラフタイプにコンストレインツを設定する事で、モデル作成時の制約を設ける事が出来ます。



モデルの作成

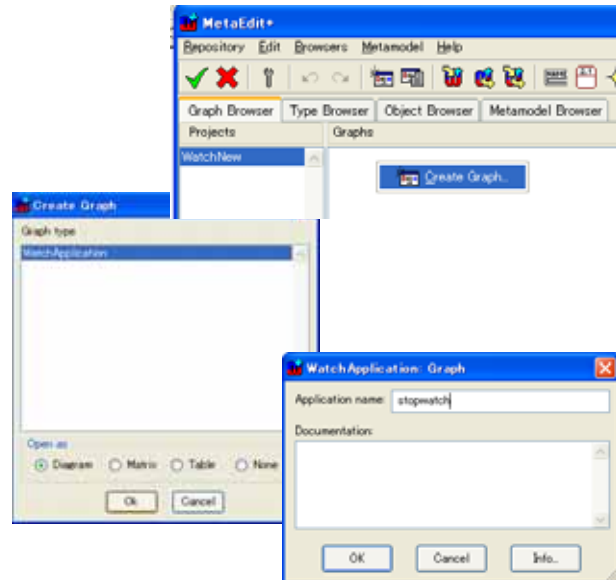
次にモデルを作成します。ここではストップウォッチのアプリケーションを作成します。



Graphの作成

WatchExampleで作業

- GraphBrowserのGraphsを右クリックして、プルダウンメニューからCreateGraphを選択
- WatchApprication GraphTypeを選択してOKボタンをクリック
- AppricationNameにstopwatchを入力してOKボタンをクリック
- WatchExampleのstopwatchを参考にモデルを作成



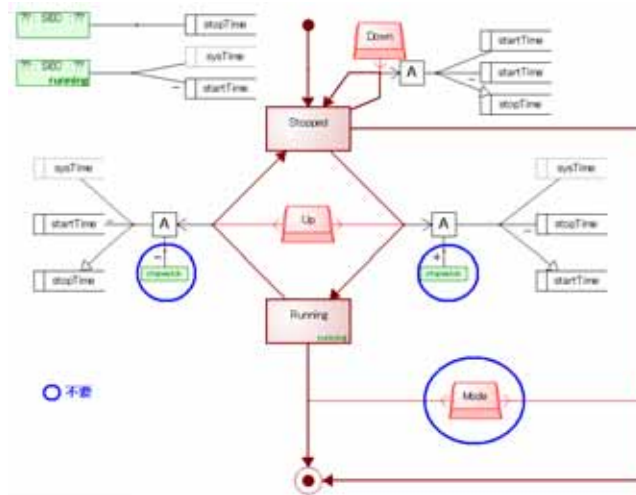
新規にグラフを作成する手順です。WatchExampleのstopwatchを参考にモデルを作成します。



Stopwatchモデルの作成

WatchExampleで作業

- 以下のモデルを参考にstopwatchモデルを作成



© 2007 MetaCase

50

実際のモデルを図を参考に作成します。作成の為のMetaEdit+の操作手順は、Evaluation tutorial

http://www.metacase.com/support/45/manuals/evaltut/et-Preface_.html

MetaEdit+ User's Guide

<http://www.metacase.com/support/45/manuals/meplus/Mp.html>
等を参考にしてください。



ソースコード生成環境の構築

ソースコードを生成する環境を構築する方法を紹介します。

前項で作成したモデルを元にJ A V Aのソースコードを生成する様に環境を構築します。



MERLとは

- MERLは、MetaEdit+からソースコード等出力するジェネレータを作成するための、簡単で効率的なスクリプト言語
- 以下を実現する為のコマンドを備えている
 - モデルを検索する
 - モデルエレメント(GOPRR)から情報を取り出す
 - 取り出した情報をスクリーンやファイルに出力する
 - 多様なユーザー割り込みや外部プログラムの実行等の追加の機能
- Generatorは、常に1つのGraphTypeと結びついている

MetaEdit+では、ソースコードやドキュメントを生成する為に、MERLと言うスクリプトを使ってモデルを巡回・検索しながら情報を取り出し、出力する事でソースコードやドキュメントの生成を行っています。MERLで書かれたスクリプトをジェネレータと呼んでいます。



MERLの基本

■ report定義の基本構造

- 例: graph内の全てのobjectを検索し、そのidを改行と共に出力する

```
Report 'Simple report'
foreach .()
{
  id;
  newline;
}
endreport
```

Diagram illustrating the structure of a report definition in MERL. The code is annotated with red arrows and brackets:

- `Report 'Simple report'` is labeled as *Generator header*.
- The `foreach .()` block is labeled as *Generator code*.
- Inside the `foreach` block, `id;` and `newline;` are grouped together as *Block*.
- The `endreport` statement is labeled as *Generator end*.

ジェネレータの基本構造は、MERLの各コマンドや出力すべき文字列をReport/endreportで囲んだ形式になっています。



ジェネレータの作成と編集

WatchExampleで作業

■ 新たなジェネレータを作成する

- Generator Editor で、Generator | Newを選択
- 或いは、編集領域で直接以下の様に入力する:

```
Report 'Report_name'
```

```
endreport
```

■ レポートの定義を編集する

- Concept Boxで適当なタイプが選択されている状態で、Choice Boxでモデリング言語のコンセプト(GOPRR)をダブルクリックする
- Concept Boxで*Templates*が選択されている状態で、Choice Boxで言語テンプレートをダブルクリックする
- 全てマニュアルで記述する

■ 編集完了後は保存を忘れない

- 保存しないと修正内容は破棄される

ジェネレータを作成するには、ジェネレータエディタを使用します。グラフのGraph -> Edit Generatorsメニューでエディタが開きます。エディタからメタモデルにアクセスする為のテンプレートが用意されており、それらを使用してジェネレータが簡単に作成出来ます。



FocusとNavigate

- Generator
 - メタモデルに照らして、モデルの中を順番に調べてゆく
 - モデルの中の必要なデータにアクセスし展開する
 - そのデータを、あらかじめ定義されているルールや意味に照らして、コードに変換して行く
 - ユーザー定義の出力フォーマットで出力できる
- Generator**実行中**
 - 注目しているgraphのコンセプト(GOPRR)のみが参照される
 - 注目しているgraphは、実行中に変わる事がある
(例、decomposition/explosionリンクを追跡した場合)
 - Focusは常に注目しているコンセプト(GOPRR)上にある

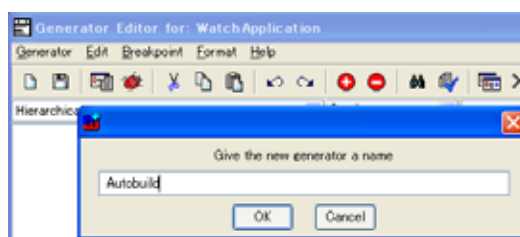
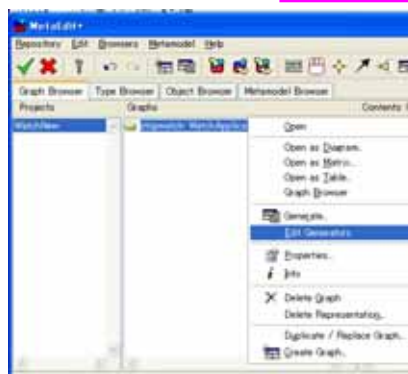
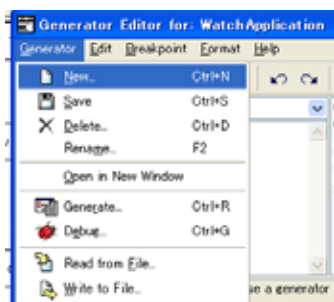
ジェネレータを実行した場合の動作を説明しています。



Autobuildの作成

WatchExampleで作業

- Graphsでstopwatchを右クリック->EditGeneratorsを選択
- GeneratorEditorでGenerator->Newを選択
- ダイアログでAutobuildを入力



実際に作成したモデルのジェネレータを作成します。ここではAutobuildという名前のジェネレータを作成しています。



バッチファイルとHTMLファイルを生成

- 生成されたソースコードのビルドと実行を行うためのバッチファイルを生成する
- ビルドされたJAVAプログラムを動作させる為のHTMLファイルを生成する
- 作成したバッチファイルを実行する

```
filename: 'reports\ak\StopWatch.bat'; write:
"cd reports; newline;
@if exist *.class del *.class
@echo on
c:\java\jdk\bin\javac *.java
start /w StopWatchTest.html; newline;
close:

filename: 'reports\ak\StopWatchTest.html'; write:
";
<html><head><title>stopwatch Watch Test Environment</title></head>
";
<body><center>
";
<h1>stopwatch</h1><h2>Watch Test Environment</h2>
";
<applet code="Juststopwatch.class" width=250 height=250</applet>
";
</center></body></html>
";
close:

external: 'reports\ak\StopWatch'; execute:

endreport
```

ジェネレータの中で、File name [ファイル名] write と記述すると出力先を指定出来ます。その後のMERLのコマンド以外(プロパティーや固定文字列)はここで指定したファイルに出力されます。Closeで出力先を閉じる事が出来ます。この仕組みを使って、コードをコンパイルするバッチファイルとコンパイルされたプログラムを実行する為のHTMLファイルを作成しています。図のコードを、先ほど作成したAutobuildジェネレータに追加します。



JAVAファイルの作成

WatchExampleで作業

- クラスの宣言
 - 変数の宣言
 - 初期化関数の定義
- State、Transition、Displayの登録

```
Report 'Autobuild'  
  
filename 'reports%stopwatch.java' write;  
'public class stopwatch extends AbstractWatchApplication {  
';  
subreport; '_Variables'; run;  
    public stopwatch(Master master) {  
        super(master, "", oid; "");' newline;  
subreport; '_StateData'; run; newline;  
subreport; '_TransitionData'; run; newline;  
subreport; '_StateDisplayData'; run;  
'    };' newline;
```

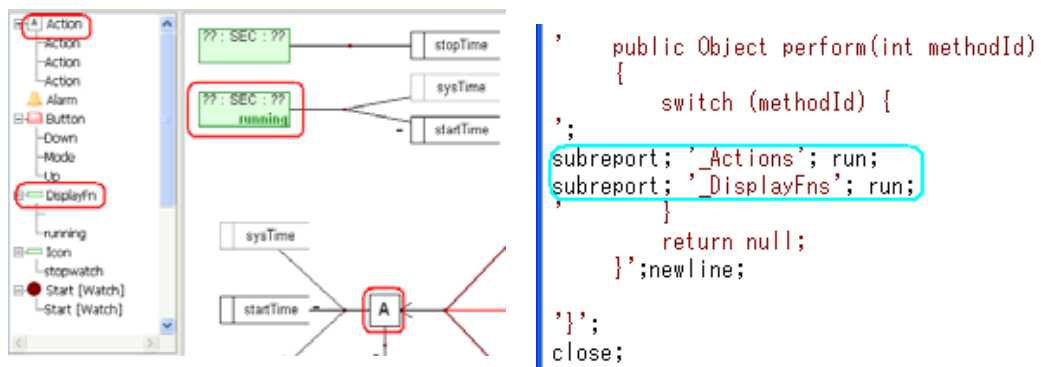
アプリケーションの実体を作成する為のスクリプトです。このスクリプトでは、各々の機能を実現するためのサブレポートを実行しています (subreport [レポート名] run)。Autobuildジェネレータの先頭部分に、図のコードを追加します。



動作に関する定義

WatchExampleで作業

- ActionオブジェクトとDisplayFnsオブジェクトの動作を司っている
- ボタンによるStateの推移は、JavaVMが行っているため、初期化時の登録のみでよい
- Action、DisplayFnsを検索するサブレポートを呼び出している



© 2007 MetaCase

59

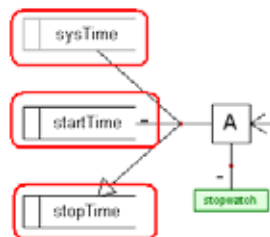
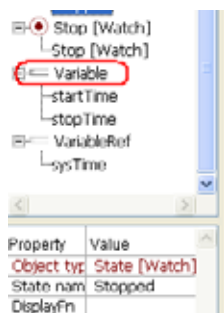
ストップウォッチの時間表示と時間計算を行う処理を記述しています。先ほど入力したコードの後ろバッチファイルとHTMLファイルを生成するコードの前に追加します。



サブレポート - 変数の宣言

WatchExampleで作業

- _Variablesレポートを新規作成
- 変数を扱うオブジェクトを宣言
- Graph内の全ての変数を検索して宣言
- Graph内の全ての変数を検索してアクセス関数を宣言



```
Report _Variables
foreach .(DisplayFn | Action)
[
  static final int ;
  if type; = 'Action' then 'a'; else 'd'; endif;
  oid; = ;
  foreach .(DisplayFn | Action)
  [
    '+';
    if oid; = oid;l; then ' //'; endif;
  ]
  newline;
]
foreach -Variable
[
  public ' ; :Type; ' ; id; ' = new ' ; :Type; '();'; newline;
];
foreach .Variable
[
  subreport; "_getSet"; run;
];
newline;
```

© 2007 MetaCase

60

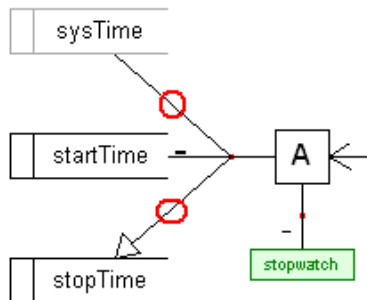
モデル内に変数オブジェクトが見つかった場合の処理スクリプトです。Autobuildジェネレータを作成した時と同様の手順で、**_Variablesジェネレータを作成してください。図のコードをジェネレータに設定してください。**



サブレポート - アクセス関数の宣言

WatchExampleで作業

- _getSetレポートを新規作成
- set[変数名]、get[変数名]の関数を宣言



```
Report '_getSet'  
' public ; :Type; ' get'; id; '() {  
    return ; id; '  
}  
public void set'; id; '('; :Type; ' t1) {  
    ; id; ' = t1;  
}  
' ;  
endreport
```

変数にアクセスする為のソースコードを生成するジェネレータの定義です。

**_Variablesジェネレータと同様の手順で
新規にジェネレータを作成し、コードを設定
してください。**



サブレポート - Stateの登録

WatchExampleで作業

- _Statedataサブレポートを新規作成
- 全てのStateをJava実行環境に登録
- Graph内の全てのStateを検索して、登録関数を記述している



```
Report '_StateData'  
  
foreach .(State [Watch] | Start [Watch] | Stop [Watch]);  
{ '    addStateOp(""; id; '", ""; oid; '"');'; newline;  
}  
  
endreport
```

© 2007 MetaCase

62

モデル内にStateオブジェクトが見つかった場合の処理スクリプトです。

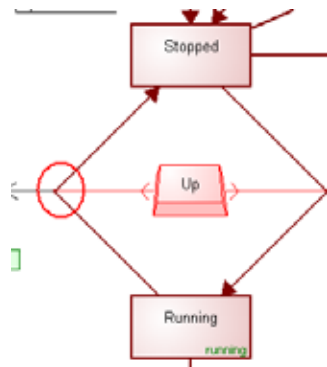
_Variablesジェネレータ同様の手順で新規にジェネレータを作成し、コードを設定してください。



サブレポート - Transitionの登録

WatchExampleで作業

- _TransitionDataサブレポートを新規作成
- 全てのTransitionをJava実行環境に登録
- Graph内の全てのTransitionを検索して、登録関数を記述している



```
Report '_TransitionData'
foreach >Transition
{
    'addTransition (';
    '"; do ~From.() {id}; '"; '";
    '"; do ~Event.() {id}; '"; '";
    if ~Action.() then
    do ~Action.() {'a'; oid};
    else
    '0';
    endif;
    '";
    '"; do ~To.() {id}; '";
    ');'; newline;
}
endreport
```

モデル内にTransitionリレーションシップが見つかった場合の処理スクリプトです。

_Variablesジェネレータ同様の手順で新規にジェネレータを作成し、コードを設定してください。



サブレポート - StateDisplayのデータ

- _StateDisplayDataContentサブレポートを新規作成
- 各StateDisplayへの表示項目を抽出

```
Report '_StateDisplayDataContent'  
  
    ', METime.'; :Central;  
    ', d'; oid;  
  
endreport|
```

ストップウォッチに表示する項目を抽出するコードを生成する為のジェネレータです。

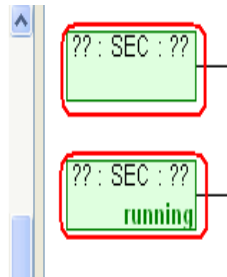
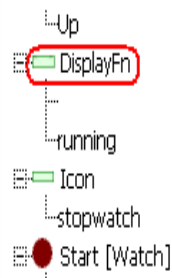
_Variablesジェネレータ同様の手順で新規にジェネレータを作成し、コードを設定してください。



サブレポート - DisplayFnsの表示

WatchExampleで作業

- _DisplayFnsサブレポートを新規作成
- DisplayFnsへの値の表示
- Graph内の全てのDisplayFnsを検索し、表示値を返している



```
Report '_DisplayFns'  
  
foreach .DisplayFn  
{  
  ' case d'; oid; ':'; newline;  
  do >()  
  {  
    ' return ';  
    subreport; '_calcValue'; run;  
    ' '; newline;  
  };  
}  
  
endreport
```

ストップウォッチに時間を表示する為のソースコードを生成する為のジェネレータです。

_Variablesジェネレータ同様の手順で新規にジェネレータを作成し、スクリプトを入力してください。



サブレポート - 値の計算

WatchExampleで作業

- _CalcValueサブレポートを新規作成
- Relationship、Roleに基づく値の計算
- 接続されている変数を全て検索し、Roleに基づいて計算するコマンドを生成している

```
Report '_calcValue'  
  
do ~Get.  
{ 'get' id '()' }  
}  
do ~(Minus|Plus)  
{ '.me' type  
  do .  
  { '(get' id '()' )  
  }  
}  
  
endreport
```

ストップウォッチの経過時間をシステムタイマーから計算するソースコードを生成するスクリプトです。

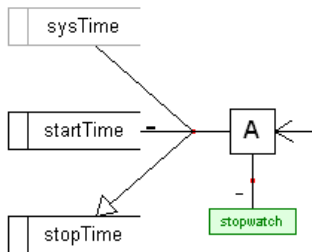
_Variablesジェネレータ同様の手順で新規にジェネレータを作成し、スクリプトを入力してください。



サブレポート - Actionの定義

WatchExampleで作業

- _Actionサブレポートを新規作成
- Actionに基づく処理の記述
- Graph内の全てのActionを検索し、ActionBody Roleで接続されている処理を行う(実際にはset Roleのみで、サブレポートを呼び出している)



```
Report '_Actions'  
foreach .Action  
{  
  ' case a'; oid; ':'; newline;  
  do ~ActionBody  
  { do >() { ' '; subreport; '_'; type; run; };  
    newline;  
  };  
  ' return null;'; newline;  
}  
endreport
```

モデル内にActionオブジェクトが見つかった場合の処理スクリプトです。

_Variablesジェネレータ同様の手順で新規にジェネレータを作成し、コードを設定してください。



サブレポート - Setの定義

WatchExampleで作業

- _Setサブレポートを新規作成
- Actionに接続された変数処理の記述
- Setに接続された変数に値をセットするために記述が書かれている

```
Report '_Set'  
  
'set'; do ~Set.() {id;}; '(';  
subreport; '_calcValue'; run;  
'');'  
  
endreport
```

変数に値をセットする為のソースコードを生成する為のスクリプトです。

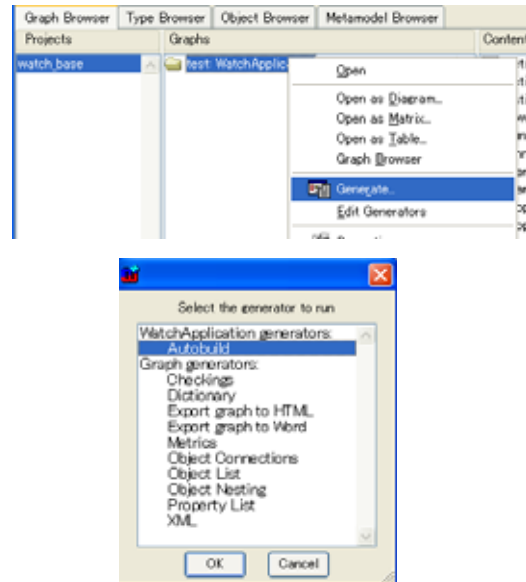
_Variablesジェネレータ同様の手順で新規にジェネレータを作成し、コードを設定してください。



ソースコード自動生成とプログラム実行

WatchExampleで作業

- GraphBrowserのGraphsを右クリックして、Generateを選択
- Autobuildを選択してOKボタンをクリック
- 新たに作成したモデルに基づいたコードが生成され、実行されます。



全てのジェネレータのセッティングが完了しました。

実際に上手くソースコードが生成され、プログラムが実行されるかを確認しましょう。



お問い合わせ

富士設備工業株式会社
電子機器事業部

E-MAIL: info@fuji-setsu.co.jp
www.fuji-setsu.co.jp

