著作権

著作権表示

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、または サービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の 製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの 評価および検証は、お客様の責任で行っていただきます。

IBM 特許権

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムと その他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、製造元に連絡してください。

Intellectual Property Dept. for Rational Software|
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

保証の不適用

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示 もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

機密情報

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

追加の法的通知が、本書で説明するライセンス付きプログラムに付随する「プログラムのご使用条件」に含まれている場合があります。

サンプルコードの著作権

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに 準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性が あることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

IBM の商標

IBM および関連の商標については、**www.ibm.com/legal/copytrade.html** をご覧ください。これは、IBM が現在所有する米国における商標の最新リストです。以下は、International Business Machines Corporation の米国およびその他の国における商標です。

このページには、IBM が使用しているすべてのコモン・ロー商標は掲載されていません。IBM が販売している製品は多数あるため、コモン・ロー商標のうち、最も重要な商標のみを掲載しております。このページに商標が掲載されていなくても、それは IBM がその商標を使用していないということではなく、その製品が現在販売されていない、または関連する市場で、その製品が重要ではないということを意味するものではありません。

他社の商標

Adobe、Adobe ロゴ、PostScript は、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows 2003、Windows XP、Windows Vista および / またはその他の Microsoft 製品は、Microsoft Corporation の米国およびその他の国における商標または登録商標です。

Pentium は、Intel Corporation の商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

目次

97

121

289

目次

ツールの紹介

DOORS®/Analyst™ は、要件管理ツール DOORS におけるビジュアル モデリング環境です。DOORS Analyst を使えば、さまざまな要求を視覚に訴える豊富な情報で補強できるようになります。このために、ビジュアルモデリング言語である UML ベースの各種ダイアグラム、シンボル、画像が用意されています。

DOORS Analyst 4.2 は、サービス指向アーキテクチャなどの最先端ソフトウェア システムの開発を目的としたツールです。DOORS Analyst 4.2 には、UML を使用してアプリケーションのモデリングを行うための豊富な機能が備わっています。UML については UML 言語ガイド を参照してください。

DOORS Analyst の機能を最大限に活用しつつ迅速に作業を立ち上げるには、製品とともにインストールされるチュートリアルマニュアル（『UML チュートリアル』など）が役立ちます。他のドキュメントやチュートリアルの最新版はサポートページに用意されています。

さらに、以下の章にも便利な情報があります。

- 便利なショートカット キー ショートカット キーのリストです。DOORS Analyst の機能に習熟するとともにショートカットキーを使うようにすると、より速く効率的な操作が可能になります。
- ツール環境の設定 DOORS Analyst を構成管理ツールおよび要件管理ツールと統合した形でセットアップする方法について説明します。

1

DOORS Analyst 4.2 の紹介

UML

DOORS Analyst には、UML 1.x と下位互換性のある UML 2.0 を基にしたモデル駆動型のツール セットがあります。以下のダイアグラム タイプをサポートします。

- ユース ケース図
- シーケンス図
- 状態機械図
- アクティビティ図
- 相互作用概観図
- クラス図
- パッケージ図
- コンポーネント図
- 配置図
- 合成構造図
- テキスト図（UML 構文）

UML ですぐに作業を開始できるようになるために、以下の情報が役立ちます。

- モデルの操作
  モデル ベースの開発の基礎について説明します。手順と概要情報を提供します。
- UML 言語ガイド
  UML 言語について説明します。
- UML チュートリアル
  サポートされるダイアグラムを使用して作業を行うためのチュートリアルです。作成したモデルをベリファイする方法についても学びます。

- UML クイック リファレンス ガイド
  グラフィックとテキストの UML で一般的に使用されている構成要素の例を示します。

DOORS における DOORS Analyst コマンド

DOORS Analyst は、DOORS 要件データベースから UML モデルのハンドリングを行うための、一連の UML ツールです。

一般的な機能

すべてのフォーマル モジュールには、UML モデルの操作を行うための一連のコマンドを持つ［Analyst］メニューが表示されます。フォーマル モジュールで［Enable Analyst］を実行すると、他のコマンドが使用できるようになります。フォーマル モジュール内の要素にも、DOORS Analyst 固有の一連のコマンドがショートカットメニューとして追加されます。オブジェクトを右クリックして選択するコマンドにカーソルを合わせると、これらのコマンドを適用できます。

UML Kind

モジュールで DOORS Analyst を有効にすると、UML の種類を表示する［Analysis Type］カラム（旧バージョンでは［Object Type］カラム）が挿入されます。先頭のカラムには、UML アイコンも表示されます。

UML ダイアグラム シンボルが同期されている場合、［Analysis Type］カラムには［Other］が表示されます。

［Analysis Type］の値が適切でない場所に非同期オブジェクトを移動すると、UML アイコンの左上に小さな赤い 感嘆符が付けられます。このマーカはオブジェクトがコンテキストから外れていることを示します。

ダイアグラム中での DOORS 属性値の表示

オブジェクトの属性値をダイアグラム中で表示できます。このために、オブジェクトを表現しているシンボルに接続されたコメントシンボルを使用します。属性は 2 つのカテゴリに分類できます。

- オブジェクト テキスト
- ユーザー定義のものを含めた、他のすべての属性

ダイアグラム中で表示する方法はそれぞれの項目を参照してください。

- オブジェクト テキスト
- 属性

注記

DOORS Analyst では、ダイアグラムはコメントをもつことはできません。したがって、ダイアグラムオブジェクト用にオブジェクトテキストやその他の属性を表示することはできません。どのシンボルにも接続しない、ダイアグラム上に配置されたコメントシンボルは、ダイアグラムとではなく所有者に関連付けられます。

オブジェクト テキスト

任意のオブジェクトのオブジェクトテキスト属性の値は、デフォルトでは DOORS とモデルに格納されます。DOORS Analyst 内で表示と編集が可能です。

ダイアグラム内でオブジェクトテキストを表示するには、以下の手順を行います。

- テキストを表示したいオブジェクトを表すシンボルを選択する
- シンボルを右クリックし、［表示 / 非表示］から［コメントの表示］を選択する

コメントシンボルを削除するには、シンボルを選択して［削除］をクリックします。コメントはダイアグラムから削除されますが、モデルからは削除されません。上の手順を再実行して再表示できます。

DOORS Analyst でオブジェクトテキストを編集するには、以下の手順を行います。

- オブジェクトテキストがコメントシンボル内に上で説明したように表示されていることを確認する
- テキストを編集する。ヘディング `Object Text` はそのままにする

オブジェクトテキスト属性の内容は、デフォルトでは同期化の間に DOORS と DOORS Analyst にプロパゲートされます。この内容はモデル内のコメントと同様に保存され、`Object Text` というヘッダーで示されます。オブジェクトテキストの同期は、UML コメント シンボル属性によって制御され、オン / オフを切り換えられます。

DOORS Analyst 内のオブジェクトに新たにオブジェクトテキストを追加するには、以下の手順を行います。

- 正しいオブジェクトを表しているシンボルがあることを確認する
- コメントシンボルを作成し、他のシンボルと接続する
- コメントシンボルの最初の行として、`Object Text` と入力する
- 次の行以降にオブジェクトテキストを入力する

同期化の間に、テキストは対応する DOORS オブジェクト内にオブジェクトテキストとして挿入されます。

UML コメント シンボル属性

オブジェクトテキスト属性の内容は、デフォルトでは DOORS と DOORS Analyst の間でプロパゲートされ、モデル内のコメントとして保存されます。

オブジェクトレベル属性「UML Comment Symbol」の値が、オブジェクトテキストをプロパゲートするかどうかを制御します。

「UML Comment Symbol」が True（デフォルト）に設定されていると、オブジェクトテキストはツール間でプロパゲートされます。「UML Comment Symbol」が False に設定されていると、オブジェクトテキストはツール間でプロパゲートされません。この値はいつでも変更可能です。

DOORS Analyst からオブジェクトテキストを表す UML のコメント シンボルを削除しても、元の DOORS オブジェクトテキストは削除されません。この場合、ツールは属性を False に設定して、プロパゲートが行われないことを示します。

属性

オブジェクトの DOORS 属性値を、DOORS Analyst 内のダイアグラムのコメントシンボルとして表示できます。

ダイアグラムで属性を表示するには、以下の手順を行います。

- DOORS で、ダイアグラム中で表示したいオブジェクトを選択する
- Analyst メニューから［Select Attributes to Show in Analyst］コマンドを実行する
- 確認したい属性を選択する
- オブジェクトを右クリックして、［Edit in Analyst］を選択する

同期後、UML コメントシンボルが DOORS オブジェクトに対応するダイアグラムシンボル用に作成されます。このコメントシンボルのヘッダーは `Attributes` であり、続いて以下の構文にしたがった属性名と属性値があります :

```
attribute name : attribute value
```

注記

ダイアグラム内に存在する前に要素に表示する属性を追加した場合は、要素をダイアグラムに追加しても、ダイアグラムにコメント シンボルは表示されません。ショートカット コマンド［コメントの表示］（［表示／非表示］のサブメニュー）を使用すると、コメントが表示されます。

DOORS Analyst のダイアグラム

フォーマル モジュールは、UML 情報をそのオブジェクトに格納します。DOORS Analyst ウィンドウが開くと、DOORS Analyst のダイアグラム エディタから情報にアクセスできるようになります。

DOORS でのモデルの保存

DOORS Analyst がセッションを終了すると、この情報は DOORS モジュール内に、UML データ モデル（拡張子 `.u2` が付いたファイルに対応）として保存されます。次回 DOORS Analyst を開くと、UML 情報はこのデータ モデルとフォーマル モジュールへの変更に従って表示されます。

複数モジュールからの要素の参照

Analyst を有効にしたモジュールでは、他の Analyst モジュールから要素を参照することが可能です。モジュールで［Edit in Analyst］を使用すると、同じダイアグラム エディタに要素が表示されるようにます。［モデル ビュー］を表示し、要素を参照するダイアグラムに［モデル ビュー］から要素をドラッグします。

共有編集モード

DOORS Analyst は、DOORS の共有編集モードをサポートします。このモードでは、複数ユーザーが同じ DOORS Analyst モジュールの別部分で同時に作業できます。

この機能を使用するには、以下の手順を行います。

- まずフォーマル モジュールで DOORS Analyst を有効にする必要があります（モジュールが排他編集モードになっている場合）。
- 通常の DOORS コマンドを使用して編集可能セクションを設定できます。詳細な手順については、DOORS のユーザーマニュアルを参照してください。
- 共有編集モードでモジュールを開きます。

  共有編集モードでモジュールを開くと、コマンド［Enable Analyst for Section］が使用可能になります。
- Analyst セクションを作成するには、ます任意の名前の DOORS オブジェクトを作成します。続いて、このオブジェクトを選択し、［Enable Analyst for Section］を選択します。

  これで、このオブジェクトの［Analysis Type］を［Model］に設定できます。これは、Analyst Type が UML モデルに対応しており、このオブジェクトに対して対応する UML のルートパッケージが作成されることを意味します。
- ［Insert UML］を使用してダイアグラムに UML オブジェクトを追加できるようになります。
- ダイアグラム エディタを開くには、［Edit in Analyst］を使用します。

  エディタを起動する前に、このセクションのすべてのオブジェクトをロックできるかチェックされます。正常に起動すると、オブジェクトがロックされ、エディタが開きます。起動できなかった場合は、エラー メッセージが表示されます。

注記

フォーマル モジュールでの Analyst モデルの階層は「フラット」でなければなりません。つまり、Analyst モデルの下のオブジェクト階層に他の Analyst モデルを配置することはできません。したがって、フォーマル モジュール内で Analyst モデルを移動する際は、他のオブジェクトの下の階層に移動しないよう十分な注意が必要です。この状況は、Analyst セクションで［Edit in Analyst］を実行するまで、つまり実際のエラーが発生するまで、検出されません。

Analyst セクションのオブジェクトが別の Analyst セクションのオブジェクトを参照できます。詳細については、複数モジュールからの要素の参照を参照してください。

Analyst の共有セクションは、「標準の」Analyst を有効にしたモジュールとともに使用することはできません。

リンク

UML 要素とのリンクは、DOORS Analyst でも DOORS と同じように赤とオレンジの矢印で示されます。モデルビュー内の矢印を右クリックすると、オブジェクトのリンク端を示すメニューが表示されます。いずれかのメニュー項目を選択すると、このオブジェクトを含む DOORS フォーマル モジュールが、オブジェクトが選択された状態で表示されます。

制限事項

コメント

DOORS Analyst が他のモジュール／セクションの要素を参照している場合、DOORS Analyst のダイアグラムを編集すると、直接編集できないセクションにある要素に影響を及ぼします。この結果、矛盾するプレゼンテーションが作成されることがあります。これは、クラス図を編集して他のモジュール内のクラスにコメントを追加したとき、そのモジュールが現在編集できない状況にあった場合などに発生します。

DOORS Analyst の属性

UML Kind、UML Location、UML Name などの属性は、内部で使用するためのものです。これらの属性を削除すると、DOORS Analyst のデータが失われます。これらの属性を削除した場合、「元に戻す」操作では回復することはできません。モジュールのベースラインがない場合は、データは完全に失われます。

Diagram/Diagram below

ダイアグラムはダブルクリックで開くことができます。これは DOORS 7.1 ではサポートされません。

Import Partition and Clone

DOORS の「Import Partition」および「Clone」機能は、現在 Analyst ではサポートされていません。

DOORS の［Copy］、［Paste］、［Paste and update references］などのコマンドを使用して Analyst モジュールをコピーした場合、元の UML 要素とコピーされたモジュールの間に競合が発生することがあります。元のモジュールとコピーされたモジュールを同時に使用する場合、このコピーの操作は行わないでください。

複数の DOORS サーバー

DOORS Analyst は複数の DOORS サーバーを同時並行的にアクセスする DOORS クライアントをサポートしません。ある DOORS データベースから［Edit in Analyst］で DOORS Analyst モジュールを開いて、さらに別の DOORS データベースから別の DOORS Analyst モジュールを開くような操作は推奨しません。

［Analyst］メニュー

Enable Analyst / Disable Analyst

このコマンドは、DOORS Analyst モードのアクティブまたは非アクティブの表示と、モードの切り替えに使用します。モジュールでメニュー項目［Enable Analyst］を選択すると UML モデルの保存が可能になり、メニュー項目［Disable Analyst］を選択すると UML モデルの保存ができなくなります。［Disable Analyst］の実行後はオブジェクトを Analyst 内で開くことはできません。

Enable Analyst for Section

このコマンドは、DOORS の共有編集モードで作業中に使用できます。Analyst の編集可能セクションを作成するには、DOORS モジュールで Analyst を排他編集モードで有効にしておく必要があります。

モデル（.u2）ファイルが DOORS モジュール内のアイコンで示されます。

Insert UML

このコマンドを使用すると、DOORS Analyst を有効にしたモジュールでダイアグラムと要素の作成が可能になります。サブメニューで UML ダイアグラムの作成と要素の作成のいずれかを選べます。また、現在の選択と同じスコープ内に新しいエンティティを作成するか、現在の選択の下位に作成するかを選べます。

Element/Element below

このコマンドを使用して、モデル要素を作成できます。作成できるモデル要素は以下のとおりです。

- アクター
- 属性
- クラス
- コンポーネント
- ノード
- パッケージ
- サブジェクト
- システム
- ユース ケース

作成されたモデル要素は、アイコンによって可視化されます。

Diagram/Diagram below

このコマンドを使用して、ダイアグラムを作成できます。ダイアグラムをダブルクリックすると、DOORS Analyst 内でダイアグラムが表示されます。DOORS Analyst を終了すると、ダイアグラムのイメージがフォーマル モジュールに保存されます。

フォーマル モジュールでは、すべてのタイプのダイアグラムがイメージとして表示されます。ダイアグラムは Windows メタファイル（拡張子 `.wmf`）として保存されます。

注記

フォーマル モジュールでは、ダイアグラムは 1 つの DOORS オブジェクトおよび 1 つのテーブル（イメージとして表示される）として表現されます。DOORS Analyst はこのテーブルをダイアグラム オブジェクトの直下のレベルに配置されていると仮定しています。ただし、他のオブジェクトもダイアグラム オブジェクトの直下のレベルに配置されることがあります。一般的には、テーブルを関連するオブジェクトの直下に続けて配置することを推奨します。

Edit in Analyst

このコマンドにより、DOORS Analyst の適切なエディタを使用してダイアグラム内の選択したモデル要素を編集できるようになります。［Edit in Analyst］は、モデル要素またはダイアグラム画像を右クリックし、表示されたショートカット メニューから選択することもできます。要素が複数のダイアグラムにある場合、そのうちの 1 つだけが表示されます。

また、このコマンドにより、フォーマル モジュールで要素を選択せずにダイアグラムを開いて編集することもできます。この場合、DOORS モジュールに対応する UML ルート パッケージが「選択された」と見なされ、コマンドのベースとして使用されます。

［Enable Analyst for Section］コマンドを適用したオブジェクトは共有編集モードで編集できます。

DOORS モジュールまたは他の共有セクションに格納されたモデル要素への参照がある場合は、Analyst のダイアグラム エディタはその要素を読み取り専用モードでロードします。このモードは、対応する DOORS モジュールを開いてこの要素に［Edit in Analyst］を選択することで、読み取り／書き込み可能に変更できます。

Select Attributes to Show in Analyst

このコマンドは、選択した DOORS 要素を DOORS Analyst に表示する際、コメント シンボルに DOORS 属性を表示するよう選択できるダイアログを開きます。

Update Diagrams

このコマンドにより、DOORS モジュールで UML 要素に行った変更を反映してすべてのダイアグラムを更新します。

Convert Module

このコマンドにより、旧バージョンの Analyst で作成したモジュールを新バージョンに更新します。

注記

Analyst で［Edit in Analyst］を使用して UML オブジェクトまたはダイアグラムを開き、それを保存した場合、ダイアグラムなども更新されます。

旧バージョンで作成された DOORS Analyst モジュールを開く場合、新バージョンで新しい属性が追加されている場合など、モジュールの変換が必要となることがあります。変換が必要かどうかは、DOORS Analyst モジュールを開くときに自動的に自動検出されます。DXL Interaction スクリプトが実行され変換が必要であることを知らせるメッセージが表示されます。［Convert Module］コマンドによって、変換が実行され、その後モジュールとダイアグラムが保存されます。

DOORS Analyst モジュールで「共有編集モード」を使用していた場合、すべてのセクションで、1 度の操作で新しいフォーマットへの変換をする必要があります。このためには、「排他編集モード」でモジュールを開く必要があります。

Help

このコマンドにより、DOORS Analyst のヘルプ ファイルを開くことができます。

About Analyst

このコマンドにより、DOORS Analyst の［About Analyst］ダイアログを開くことができます。このダイアログにはツールのバージョンとライセンス情報が表示されます。

DOORS Analyst のダイアグラム表示

DOORS Analyst が開くと、フォーマル モジュールのダイアグラムから情報にアクセスできるようになります。

ここで行った変更は、DOORS Analyst で保存を行うとフォーマル モジュールに反映されます。

重要！

DOORS と DOORS Analyst 間を移動する場合は、常に、［Edit in Analyst］を使用するか、ダイアグラムをダブルクリックします。DOORS Analyst から DOORS へ移動する場合、必ず［Analyst］ウィンドウを閉じるか、ショートカット コマンド［Edit in DOORS］を使用します。これで要素間の同期を適切に取ることができます。また、DOORS のウィンドウに移動する前に DOORS Analyst で明示的に保存を行うこともできます。

DOORS Analyst ユーザー インターフェイスの概要

基本レイアウト

DOORS Analyst のユーザー インターフェイスには、**デスクトップ**とツールバーの 2 つの領域があります。さらに、フレームの下部にはステータス バー、インターフェイス フレームの上部にはメニュー バーとツール バーの領域があります（基本レイアウトではメニューは表示されません）。

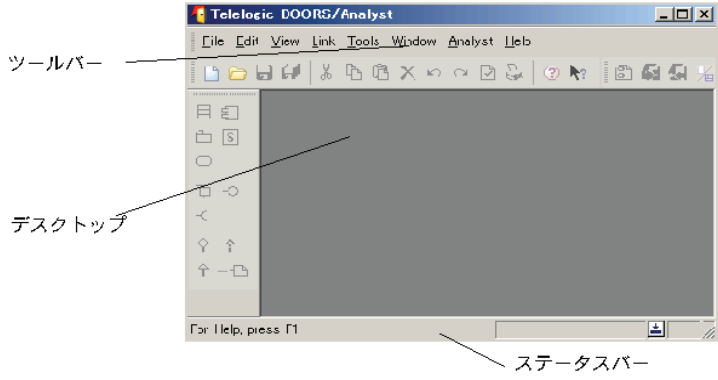

図 1: DOORS Analyst の基本レイアウト

詳細レイアウト

全体の DOORS Analyst ユーザー インターフェイスには、**デスクトップ**、**ワークスペース** ウィンドウ、**ショートカット** ウィンドウ、および出力ウィンドウがあり、必要に応じて表示／非表示を切り替えられます。さらに、フレームの下部にはステータス バー、上部にはメニュー バーとツール バーがあります。ウィンドウはすべて必要に応じてドッキングできます。よく使用するツールバーをショートカット ウィンドウにドラッグアンドドロップすることもできます。

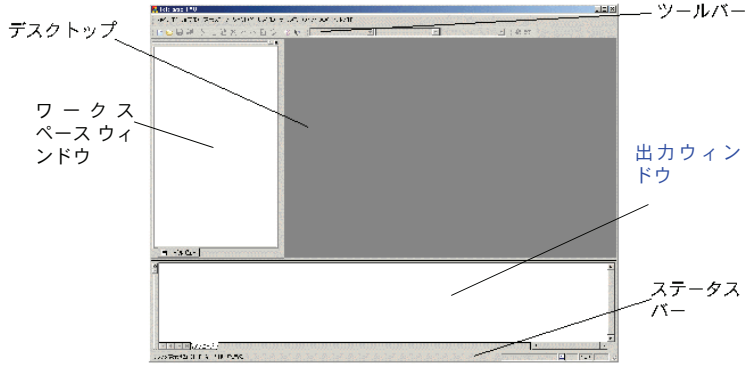

図 2: DOORS Analyst のデスクトップ

デスクトップ

デスクトップは編集領域ともいい、作業中のドキュメントが表示される場所です。ここで実際の開発を行います。編集または表示のために開かれたダイアグラム、ドキュメント、ソース ファイルは、デスクトップに表示されます。表示されるエディタまたはビューアのタイプは、プロジェクトに含まれるファイルのタイプによって異なります。

デスクトップに 2 つ以上のドキュメントが開かれている場合、［ウィンドウ］メニューのコマンドを使用するか、Ctrl + Tab キー（前面へ）または Ctrl + Shift + Tab キー（背面へ）を押してウィンドウ間を移動できます。

ヒント

エディタを全画面表示にするには、［表示］メニューから［全画面表示］を選択します。元の画面表示に戻すには、Esc キーまたは Alt + 1 キーを押します。

参照

ウィンドウの操作

ワークスペース ウィンドウ

ワークスペース ウィンドウは、ワークスペース情報の構造を複数のビューとして表示し、管理するためのグラフィック ツールです。

ワークスペース ウィンドウには、展開可能なノードを含んだ情報構造が表示されます。これらのノードを展開／折りたたんだり、または、他のビューを使用することにより、情報を必要な分に絞り込んだ形でワークスペースに表示できます。

ワークスペース ウィンドウを移動して、フローティング パレットとして使用できます。フローティング パレットとして使用しない場合は、左端の位置にドッキングします。ドッキング状態では、ウィンドウは水平方向にのみサイズ変更できます。上下の境界は、上はツール バー、下は出力ウィンドウによって決定されます。

ビュー

ワークスペースにはさまざまなビューを表示できます。各ビューへは対応するタブからアクセスできます。1 つのビューはモデルの 1 つの局面を表示しています。

ファイル ビュー

［ファイル ビュー］では、ファイルとして表現されるすべての要素がワークスペースに表示されます。

初めて DOORS Analyst を起動したときは、［ファイル ビュー］は表示されません。［ファイル ビュー］を表示するには、ワークスペース ウィンドウのフレーム（［モデル ビュー］タブ）を右クリックし、［ファイル ビュー］を選択します。

［ファイル ビュー］を表示するには、**ワークスペース ウィンドウ**の［ファイル ビュー］タブをクリックします。このビューで、すべてのファイルを開き、編集し、保存できます。ただし、ここでファイルを削除しても、ファイルは［ファイル ビュー］から削除されるだけで、OS のファイル システムからは削除されません。

フォルダを作成して複数のファイルを容易に管理できます。ファイルをフォルダに入れるには、そのファイルを［ファイルビュー］でドラッグ アンド ドロップします。ファイルのプロパティを表示するには、右クリックし、表示されるショートカット メニューから［プロパティ］を選択します。

モデル ビュー

［モデル ビュー］には、作業対象のすべてのデータが「抽象的な構造」として表示されます。すべての UML 要素は、このビューに表示されます。モデルに要素を追加してダイアグラムを作成するには、このビューを使用します。

このビューに表示される要素は、モデルの図形表現と見なされます。設計プロセスでは、ダイアグラム エディタを使ってもかまいませんが、［モデル ビュー］のノードについて作業するだけでも簡単にシステム全体を設計できます。

モデル ビューを表示するには、**ワークスペース** ウィンドウの［モデル ビュー］タブをクリックします。

プロジェクト ノードまたはモデル ノードのショートカット メニューには、［モデル ビュー フィルタ］というサブメニューがあります。このサブメニューをチェックすると、定義済みフィルタを［モデル ビュー］に適用できます。

メタモデルに **Resource** を基底とする 1 つのメタクラスが含まれている場合があります。このモデル要素は、実行時にロードされた要素内のリソース要素に対応付けられ、［ファイルの表示］モデル ビュー フィルタが有効になっている場合のみ［モデル

ビュー］内で表示されます。［モデル ビュー］にファイルとリソース要素を表示するには、［モデル ビュー フィルタ］ショートカット サブメニューから［ファイルの表示］を選択します。

メタモデルに Diagram のサブクラスであるオブジェクト モデル クラスを基底とするメタクラスが含まれている場合があります。このモデル要素は、ダイアグラムに対応付けられ、［ダイアグラムの表示］モデル ビュー フィルタが有効になっている場合のみ［モデル ビュー］内で表示されます。

便宜上、メタクラスは「構造」エンティティと「詳細」エンティティに分類されます。これは、［詳細の表示］フィルタを適用したときの［モデル ビュー］表示内容に影響があります。

メタモデルに Implementaion のサブクラスであるオブジェクト モデル クラスを基底とするメタクラスが含まれている場合があります。このモデル要素は、実装指向の要素に対応付けられ、［実装の表示］モデル ビュー フィルタが有効になっている場合のみ［モデル ビュー］内で表示されます。

［定義のソート］フィルタを有効にすると、そのモデル ビュー ノードの要素は辞書順でソートされて表示されます。ソート操作は、ダイアグラム ノードそのものには影響を及ぼしません。

［表示］メニューから［モデル ビューの再構成］コマンドを使用して、［モデル ビュー］に表示する情報を定義済みメタモデルに基づいて絞り込めます。この操作は、選択した要素が属するプロジェクトに影響を及ぼします。

［Standard View］以外のフィルタを選択した場合に［モデル ビュー］内のノードが「消える」ことがあります。これは、DOORS Analyst の操作に、UML 情報の格納に使用する基本的なメタモデルに依存する操作（ドラッグ アンド ドロップなど）と、現在選択されているメタモデルに依存する操作（［モデル ビュー］でツリー内の要素を表示するなど）があることに由来します。表示を［Standard View］に切り替えると、「消える」ことはありません。このモデルが基本的なメタモデルと同じだからです。一方［ダイアグラム ビュー］は、ダイアグラムを所有できるモデル要素を基本に情報を表示し、ダイアグラム ノードを所有者要素の直下に配置します。

以下の操作を行うと、この状態が発生することがあります。

- ドラッグ アンド ドロップ
- 切り取りと貼り付け
- モデル ナビゲータの［ダイアグラム］タブによるダイアグラムの作成

上のように「消えた」ノードを復元するには、以下の 2 通りの方法があります。

- ［元に戻す］機能を使用してノードを元の場所に戻す。
- 表示を［Standard View］に切り替える。この表示ではすべてのノードが表示されます。［モデル ビュー］でドラッグ アンド ドロップで移動した場所に表示されなかったノードが、新しい場所に表示されるようになります。

ショートカット ウィンドウ

ショートカット ウィンドウにツール バーを表示するように設定できます。ショートカット ウィンドウにツール バーを表示するには、ツール バーを右クリックし、ショートカット メニューから［ショートカット］を選択します。ツール バーを元の位置に戻すときは、ショートカット ウィンドウにあるツール バーを右クリックし、ショートカット メニューから［ツール バー］を選択します。

ショートカット ウィンドウの表示と非表示は、［表示］メニューの［ショートカット］コマンドで切り替えられます。

注記

一部のツール バーはショートカット ウィンドウに表示できません。

出力ウィンドウ

出力ウィンドウは、各種ツールの情報を記録、表示するための複数のタブから構成されます。たとえば、エラー メッセージ、警告、アクションの実行結果、イベントのログなどです。ツールごとに個別のタブがあります。

タブによっては、サブジェクト カラム内の要素からダイアグラム上の要素までナビゲートできるものがあります。

出力ウィンドウの表示と非表示は、［表示］メニューの［出力］コマンドで切り替えられます。

一般的なタブ

メッセージ

［メッセージ］タブには、プロジェクトの読み込みやその他の実行されたアクションに関する情報が表示されます。

このタブからツールの他の部分へのナビゲートはできません。

検索結果

このタブには、検索操作の結果が表示されます。

プレゼンテーション

このタブには、プレゼンテーションの一覧表示操作の結果が表示されます。

参照

このタブには、参照の一覧表示操作の結果が表示されます。

スクリプト

［スクリプト］タブには、スクリプトの実行結果が表示されます。

UML ツール用のタブ

チェック

モデルの完全なチェックを開始してエラーと警告を検出できます。このタブには、チェックの結果が表示されます。エラーは修正後もリストに残ります。もう一度チェック手順を実行するとこのリストが変更されます。

ナビゲート

このタブには、表形式のモデル ナビゲーション ツールであるモデル ナビゲータがあります。このツールは、モデル内のナビゲートに使用します。

ウィンドウの操作

［ウィンドウ］メニューは詳細レイアウトでのみ表示されます。

ウィンドウの配置

すべてのドキュメントウィンドウを並べて表示するには

- ［ウィンドウ］メニューから［水平方向に並べて表示］または［垂直方向に並べて表示］をクリックします。

すべてのドキュメントウィンドウを重ねて表示するには

- ［ウィンドウ］メニューから［重ねて表示］をクリックします。

ドキュメント ウィンドウを移動するには

注記

タブ付きドキュメントのあるウィンドウはドッキング状態を変更できません。

1. ドキュメント ウィンドウのタイトル バーを右クリックします。
2. メニューから以下のコマンドを選択します。
   - **ドッキング済み**：ウィンドウをアプリケーション ウィンドウ内にドッキングします。ウィンドウをドッキングする場所を選択できます。
   - **フローティング**：アプリケーションの外側でウィンドウを移動できます。
   - **MDI 子ウィンドウ**：編集領域内でのみウィンドウを移動できます。ウィンドウを最大化、最小化、縮小（元に戻す）できます。

アクティブ ドキュメントを全画面表示するには

- ［表示］メニューから［全画面表示］をクリックします。
  または
- **Alt + 1** キーを押します。

アクティブ ドキュメントを標準サイズで表示するには

アクティブ ドキュメントを全画面表示から標準サイズに戻すには、以下のいずれかを行います。

- 画面の上部にカーソルを移動します。メニュー バーが表示されたら、［表示］メニューから［全画面表示］をクリックします。
  または

- **Alt + 1** キーを押します。

ウィンドウの表示と非表示

ワークスペース ウィンドウを表示／非表示するには

- ［表示］メニューから［ワークスペース］をクリックします。
  または
- **Alt +0** キーを押します。

出力ウィンドウを表示／非表示するには

- ［表示］メニューから［出力］をクリックします。
  または
- **Alt + 2** キーを押します。

ウィンドウを閉じる

ドキュメント ウィンドウを閉じるには

- ［ウィンドウ］メニューから［閉じる］をクリックします。

すべてのドキュメント ウィンドウを閉じるには

- ［ウィンドウ］メニューから［すべて閉じる］をクリックします。

新規ウィンドウの作成

新規ドキュメント ウィンドウを作成するには

- ［ウィンドウ］メニューから［新しいウィンドウ］をクリックします。

タブ付きドキュメント

［一般］オプション ページで［タブ付きドキュメント］オプションを選択すると、1 つのウィンドウに複数のドキュメントがタブ付きで表示されます。

タブを右クリックして［切り離す］をクリックすれば、タブ付きのウィンドウからドキュメントを独立できます。独立した状態では、通常の MDI 子ウィンドウと同じように機能し、ドッキング状態を変更できます。

ウィンドウのドッキング

DOORS Analyst フレームワークには、エディタ ウィンドウに 3 種類のモードがあります。これらのモードは、ダイアグラム ウィンドウのタイトル バーを右クリックし、表示されたショートカット メニューで個々に設定できます。

注記

タブ付きドキュメントのあるウィンドウはドッキング状態を変更できません。

ドッキング済み

ドッキングされたエディタ ウィンドウは、ワークスペース ウィンドウや出力ウィンドウのように、DOORS Analyst フレームワークに沿って組み込まれます。ウィンドウを移動して適切な表示にできます。

フローティング
フローティング ウィンドウは、DOORS Analyst フレームワークの上側に配置されます。フレームワークのフレーム内に移動すると、ドッキング ウィンドウになります。

MDI 子ウィンドウ
MDI 子ウィンドウはデスクトップ領域に配置されます。この領域内で手作業で調整したり、［ウィンドウ］メニューのコマンドを使用して調整できます。

ドッキング ウィンドウの自動非表示（Windows）

グリッパ バーにピンが表示されているウィンドウは自動非表示モードに設定できます。ピンをクリックするウィンドウは非表示になり、そのウィンドウを表すラベルが表示されます。マウス カーソルをラベルに合わせると、隠れていたウィンドウが表示されます。ウィンドウをドッキングするには、ピンをもう一度クリックします。

ドッキング ウィンドウの拡大と縮小

2 つのドッキング ウィンドウがメイン ウィンドウの同じ辺を共有している場合、ウィンドウのグリッパ バー（利用可能な場合）の矢印をクリックすると、メイン ウィンドウの全面までウィンドウを拡大できます。これによって、同じ辺を共有していたもう 1 つのウィンドウは最小化されます。ウィンドウを元のサイズに戻すには、矢印をもう一度クリックします。

保存済みワークスペース ウィンドウ

セッション中に開かれていたすべてのウィンドウは、ワークスペースを再ロードすると、再び開かれます。情報は、ワークスペースと同じパスと名前で .ttx ファイルに保存されます。

参照

ビューの体系化

メニュー バーとツール バー

ワークスペースの環境設定と画面サイズにより、必要なツール バーを表示したり、まったく表示しないように設定できます。必要に応じてツール バーにコマンド ボタンを追加したり、ボタンのサイズを変更したり、別の場所に移動できます。

メニュー バー

メニュー バーには［ファイル］、［編集］、［プロジェクト］などのなじみ深いメニューがあります。実行しているタスクによって、メニューの数が変わります。

ほとんどのメニュー コマンドにはショート カットが割り当てられています。全タイプ共通のリファレンス ガイドに便利なショートカット キーのリストがあります。

Telelogic 以外のツールを簡単に使用できるようにするため、［ツール］メニューにコマンドを追加できます。この設定は、［ツール］タブを使用して行います。

たとえば、［ツール］メニューに Windows のメモ帳を追加する方法を以下に示します。

［ツール］メニューにコマンドを追加するには

1. ［ツール］メニューから［カスタマイズ］ダイアログを選択し、次に［ツール］タブをクリックします。
2. ［新規（挿入）］ボタンをクリックします。
3. ［ツール］メニューに表示したいツール名を入力し、Enter キーを押します。

   たとえば、Windows のメモ帳のコマンドを追加したい場合、「メモ帳」と入力します。
4. ［コマンド］ボックスにプログラムのパス（例：`C:¥Windows¥notepad.exe`）を参照または手動で入力します。
5. ［引数］テキストボックスに、プログラムに渡す引数を参照または手動で入力します。メモ帳アクセサリの場合、このフィールドは空のままにします。

注記

［引数］テキストボックスの隣のドロップダウン ボタンをクリックして、使用できる引数のリストを表示できます。リストから引数を選択し、［引数］テキストボックスに引数構文を挿入します。

6. ［初期ディレクトリ］ボックスに、コマンドの実行形式ファイルがあるファイル ディレクトリを指定します。メモ帳アクセサリの場合、このフィールドは空のままにします。

［ツール］メニューにコマンドが表示されたら、それをクリックしてプログラムを実行できます。

プログラムに渡す引数を［引数］テキストボックスに入力するか、プログラムの初期ディレクトリを［初期ディレクトリ］テキスト ボックスに入力して追加できます。

［ツール］メニューに追加するプログラムに `.pif` ファイルがある場合、［初期ディレクトリ］テキスト ボックスで指定されたディレクトリは、`.pif` ファイルで指定される起動ディレクトリに置き換わります。

ツール バー

ツール バーにより、頻繁に使用するツールをすばやく使用できるように、パレットに設定できます。ツール バーを変更すると、その変更は保存されて次の作業セッション時に反映されます。

標準のツール バーは、メニュー バーから使用可能な操作に対応しています。標準のツール バーは、［表示］メニューの［標準］コマンド、またはツール バー領域のショートカットメニューによって表示／非表示を切り替えられます。標準以外のツール バーはショートカットメニューによってのみ切り替えられます。

注記

一部のツール バーとコマンドは修正できません。この機能は、DOORS Analyst フレームワークに属する機能であり、エディタに関するツール バーには対応していません。

ツール バーのボタンを追加するには

1. 変更しようとするツール バーが表示されていることを確認します。
2. ［ツール］メニューから［カスタマイズ］ダイアログを選択し、次に［コマンド］タブをクリックします。
3. ［カテゴリ］ボックスでカテゴリ名をクリックし、［ボタン］領域のボタンまたは項目を表示されているツール バーにドラッグします。

ツール バーのボタンを削除するには

1. 変更しようとするツール バーが表示されていることを確認します。
2. ［ツール］メニューから［カスタマイズ］ダイアログを選択し、次に［コマンド］タブをクリックします。
3. ボタンを削除するには、ボタンをツール バーの外側にドラッグします。

デフォルトのボタンをツール バーから削除しても、［カスタマイズ］ダイアログにはそのボタンが残ります。表示をカスタマイズしたツール バーボタンを削除すると、その表示は完全になくなります。ただし、コマンドは［カスタマイズ］ダイアログの［コマンド］タブから使用可能です。

ヒント

表示をカスタマイズしたツール バー ボタンを再利用のため保存するには、未使用のボタンを格納するツール バーを作成してこのボタンを移動し、格納したツール バーを非表示にします。

ツール バーを表示／非表示するには

1. ［ツール］メニューから［カスタマイズ］ダイアログを選択し、次に［ツール バー］タブをクリックします。
2. 表示／非表示したいツール バーを、［ツール バー］リストから選択／選択解除します。
3. ［閉じる］をクリックします。

または、

1. ユーザー インターフェイスのツール バー領域の任意の場所を右クリックします。
2. 表示または非表示にしたいツール バーをクリックします。メニューは自動的に閉じます。

ツール バー ボタンの表示を変更するには

1. ［ツール］メニューから［カスタマイズ］ダイアログを選択し、次に［ツール バー］タブをクリックします。
2. 以下のオプションを選択します。
   - **ツールチップを表示**：ツール バーのボタンまたはフィールドにカーソルを移動するとツールチップが表示されます。
   - **大きいボタン**：ツール バーのボタンのサイズを大きくします。
3. ［閉じる］をクリックします。

ステータス バー

ステータス バーには、いくつかのタスクの状態に関する有意義な情報が表示されます。たとえば、エラーやツールチップなどが表示されます。また、進捗状況や現在のアクションなども表示されます。

テキスト ファイルについては、ステータス バーの右端に現在のライン番号とカラム位置が表示されます。

ラインの移動

テキスト ファイル内の指定行に移動するには、Ctrl + Shift + G キーを押し、表示されたダイアログで、移動先のライン番号を入力します。

プログレス バー

ワークスペースを開くとき、ステータスバーの右側に全進捗状況を示すプログレス バーが表示されます。

メッセージ フィールドにも、個々のロード プロセスの進捗状況を示すプログレスバーが表示されます。このとき、進行中の現在のアクションを示すメッセージも表示されます。

オプション

ツールのオプションは、現在のプロジェクトまたはワークスペースだけではなく、DOORS Analyst 全体にも影響を及ぼします。これらのオプションは、以下の方法で変更できます。

［オプション］ダイアログには、変更するオプションごとに別のタブがあります。タブの数は現在アクティブなプロジェクトのタイプによって異なります。［オプション］ダイアログのオプションの説明を表示するには、ダイアログのタイトル バーにあるクエスチョン マークをクリックし、次に目的のオプションをクリックします。

オプション ファイル

オプション設定は、オプション ファイル .tot に保存できます。このファイルは後で編集できます。

インストール時、内部フレームワークの設定とオプションを含む拡張子「.tot」の付いたファイルが多数作成されます。通常これらのファイルはユーザーが編集すべきではありません。拡張子「.tot」の付いたファイルを編集すると、データが失われたり、ツールセットが正しく使用できなくなる場合があります。オプションを変更する必要がある場合は、［ツール］メニューの［オプション］ダイアログを使用して行います。

オプションの変更

オプションを変更するには

1. ［ツール］メニューから［オプション］をクリックします。
2. ［オプション］ダイアログのタブを使用して、オプションの選択または選択解除を行います。［詳細］タブで F2 キーを押してオプションの値を指定します。
3. ［OK］をクリックします。

オプション ファイルの操作

現在のオプションを新しいオプション ファイルに保存するには

1. ［プロジェクト］メニューから［オプション］をクリックし、次に［名前を付けて保存］をクリックします。
2. ［名前を付けて保存］ダイアログで、オプション ファイル（.tot）の名前と保存場所を指定します。
3. ［保存］をクリックします。
4. アクティブなプロジェクトにオプション ファイルを含めるかの確認を求められるので、［はい］をクリックします。

モデルとダイアグラム

モデル

モデルは、作業対象のシステムを書き表したすべてのダイアグラムから構成されます。各ダイアグラムは、それぞれ、アプリケーションの特定の側面を表します。UML を使ってシステムのモデリングを行う場合、クラス図はエンティティとそのエンティティ間の関係を表現しています。

ユース ケース図およびシーケンス図で表されたユース ケースを使うと、外部との相互作用とシステムの振る舞いの概要を記述できます。

アクティビティ図と相互作用概観図を使うと、モデルにおける同時並行的な振る舞いを記述できます。

状態機械図では、アクティブなクラスの振る舞いを記述し、合成構造図では、エンティティの外部への振る舞いと他のエンティティとの相互作用を記述します。

アプリケーションはモデルを元にコンパイルされます。タイプの異なるダイアグラムには、モデルの異なるビューが表示されます。つまり、ダイアグラムで使用可能なエンティティはモデルに存在しますが、モデルに存在するエンティティが必ずしもダイアグラムで使用可能とは限らない、ということです。

モデル要素

シンボル（プレゼンテーション要素）をダイアグラムから削除しても、モデル内の対応するエンティティが削除されることはありません。同じエンティティが他のダイアグラムで使われている可能性があるからです。

しかし、モデル内のエンティティを削除すると、ダイアグラム内の対応するシンボルが削除されます。これは、アプリケーションを表現するのはモデルであり、ダイアグラムはモデルのある側面を示しているに過ぎないからです。

モデル要素と表示要素が 1 対 1 の関係にあれば、表示要素を削除するとモデル要素も削除されます。モデル要素が編集可能な表示要素を 1 つだけ持つ場合、1 対 1 の関係になります。これは、たとえば状態機械図のシンボルの場合に当てはまります。

モデル要素はワークスペース ウィンドウの［モデル ビュー］に表示されます。

参照

第 i 章「モデルの操作」

第 i 章「UML 言語ガイド」

ビュー

ダイアグラム

ダイアグラムは、UML を使ってモデルを表現したものです。ダイアグラムのタイプにしたがって、異なるプロパティとアクションを定義できます。

ダイアグラムは、通常、1 つのモデルのさまざまな見方 - ビューを表します。ダイアグラムにはさまざまなタイプがあります。ダイアグラムの名前は、UML の概念から導き出されたものです。サポートされるダイアグラムのタイプは、以下のとおりです。

- アクティビティ図
- クラス図
- コンポーネント図
- 合成構造図
- 配置図

- 相互作用概観図
- パッケージ図
- シーケンス図
- 状態機械図
- ユース ケース図

ダイアグラムの使用

モデルを表現するために、いくつかの異なるタイプのダイアグラムを使用できます。ここでは、ダイアグラムの使用方法について説明します。

UML モデルを構築する際、どのような作業手順にするかは決まっていません。26 ページの図 3 は、作業手順の 1 つの例です。

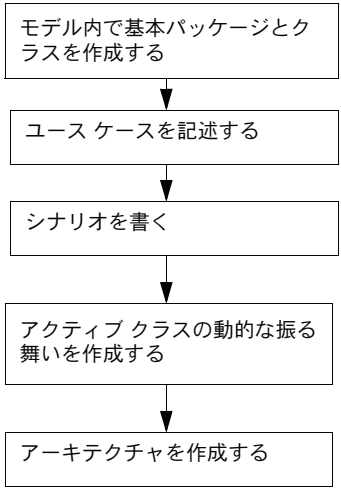

図 3: ダイアグラム作成のワークフロー

モデル内で基本パッケージとクラスを作成する

［モデルビュー］内で新規パッケージを直接作成できます。これには、パッケージを右クリックし、ショートカット メニューから［新規モデル要素］を選択し、サブメニューから目的の要素を選択します。クラス図を追加するには、［モデル ビュー］でパッケージを右クリックし、ショートカット メニューから［新規ダイアグラム］を選択し、［クラス図］を選択します。

ユース ケースを作成する

ユース ケース図はパッケージ下に直接配置するか、クラス下に配置するか、コラボレーション下にグループ化できます。ユース ケースはパッケージ、クラスまたはコラボレーションに直接挿入できます。

シナリオを書く

ユース ケースを説明するシナリオは、シーケンス図として表現すると理解しやすくなります。シーケンス図は、構文が単純かつ直感的なので、動的な振る舞いの設計の基礎にも適しています。

クラスの動的な振る舞いを作成する

次のステップは、「アクティブ」として設定されたクラスの振る舞いの定義です。この作業には、状態機械図を使用します。アクティブ クラスごとにモデルに状態機械図を追加します。その状態機械図を 開くと、アクティブツール バーで使用可能なシンボルを使用したクラスの振る舞いを定義する内部状態機械が作成されます。

アーキテクチャを作成する

次のステップは、オブジェクトのやりとりの方法を定義することです。オブジェクトのインスタンス（パート）化とパート間のやりとりは、合成構造図で記述できます。合成構造図は、クラスの内部構造、クラスの属性、およびクラスのインスタンス化を表します。

次のステップへ

さらに作業を続けてゆくには、テスト プロジェクトを作成し、DOORS Analyst を使用してさまざまなダイアグラム、要素、シンボルについての作業を行う方法について理解してください。

参照

第 i 章「UML 言語ガイド」
第 i 章「モデルの操作」

ヘルプの使い方

ヘルプ ファイルには、対応する機能についての基本的なトピックと詳細なトピックがあります。

他に利用できるドキュメントについては、第 12 章「その他のリソース」を参照してください。チュートリアル、言語解説、インストール ガイド、DOORS Analyst サポート サイトなどの外部サイトへのリンクがあります

インストール CD には、Adobe PDF 形式のドキュメントがあります。

ヘルプ ファイル内での移動

ヘルプ ファイルには、以下のように、情報を簡単に見つけるための機能があります。

- 28 ページの「検索」
- 29 ページの「検索ハイライト」
- 29 ページの「キーワード」
- 29 ページの「検索またはキーワード検索の同期」
- 29 ページの「ヘルプ ファイルのトピックのお気に入り登録」
- 30 ページの「ヘルプ トピックの印刷」

検索

完全なテキスト検索を実行するには

1. ヘルプ ビューアで［検索］タブをクリックします。
2. ［探したい語句を入力してください］フィールドに探す文字列を入力します。検索文字列として、通常表現、演算子、およびネストされた表現を使用できます。
3. また、［以前の結果から検索］、［類似する文字に合致］、および［タイトルのみ検索］のオプション（複数可）をチェックすることもできます。
4. ［検索開始］をクリックします。
5. 検索結果を表示するには、［トピックの選択］リスト内のトピックをダブルクリックするか、トピックを選択して［表示］をクリックします。

例 1: ——————————————————————————————

「link」で始まる単語を検索するには、［探したい語句を入力してください］フィールドに以下のように入力します。

```
link*
```

——————————————————————————————

検索ハイライト

検索した単語は、検索結果の全ページでハイライト表示されます。この機能は必要に応じてオフにできます。

検索ハイライトをオフにするには

1. ヘルプ ビューアで［オプション］ボタンをクリックし、メニューの［検索ハイライト オフ］をクリックします。
2. すでに検索を行っている場合、ヘルプ ビューアで［表示］ボタンをクリックすると、検索した単語がハイライト表示されなくなります。

再度オンにするまで、検索ハイライト機能はオフのままです。

検索ハイライトをオンにするには

1. ヘルプ ビューアで［オプション］ボタンをクリックし、メニューの［検索ハイライト オン］をクリックします。
2. すでに検索を行っている場合、ヘルプ ビューアで［表示］ボタンをクリックすると、検索した単語がハイライト表示されます。

再度オフにするまで、検索ハイライト機能はオンのままです。

キーワード

索引のリストを表示するには、［キーワード］タブを選択します。探している項目を見つけるには、単語の先頭文字を入力するか、リストをスクロールして検索します。項目を表示するには、項目をダブルクリックするか、項目を選択して［表示］をクリックします。

検索またはキーワード検索の同期

検索またはキーワード検索機能を使用している場合、検索結果は右側のウィンドウに表示されます。表示されているトピックが目次のどこにあるか確認するには、［同期］ボタンをクリックします。この方法で、関連トピックを見つけたり、次に検索するときのためにこのトピックが目次のどこにあるかがわかります。

ヘルプ ファイルのトピックのお気に入り登録

頻繁に参照するトピックや作業にとって重要なトピックがある場合、ウェブ ブラウザでよく行うように、「お気に入り」に登録できます。

トピックを「お気に入り」に登録するには

1. ［目次］、［キーワード］、［検索］などのタブを使用してトピックを検索します。
2. ［お気に入り］タブをクリックします。表示中のトピックの名前が［現在のトピック］フィールドに表示されます。
3. ［追加］をクリックします。追加したトピックの名前が［トピック］リストに表示されます。

ヘルプ トピックの印刷

1 つのトピック、または同じ章内の複数のトピックを選択して印刷できます。

現在表示されているトピックを印刷するには

- トピック ウィンドウを右クリックし、メニューから［印刷］を選択します。印刷ダイアログが表示されます。

目次から 1 つのトピックを印刷するには

1. トピック ウィンドウを右クリックし、メニューから［印刷］を選択します。印刷ダイアログが表示されます。
2. ［トピックの印刷］ダイアログで［選択されたトピックの印刷］をクリックし、［OK］をクリックします。印刷ダイアログが表示されます。

複数のトピックを印刷するには

1. 目次のブック アイコンを右クリックし、メニューから［印刷］を選択します。印刷ダイアログが表示されます。
2. ［トピックの印刷］ダイアログで［選択された見出しおよびすべてのサブトピックを印刷］をクリックし、［OK］をクリックします。印刷ダイアログが表示されます。

ヘルプでの検索構文

ヘルプビューアでは完全なテキスト検索を実行できます。また、文字（a-z）と数字（0-9）の組み合わせで検索文字を指定できます。「the」や「a」、「and」、「but」などの単語は制限されているので検索対象となりません。さらに、コロン（:）、セミコロン（;）、ハイフン（-）およびピリオド（.）などの句読点も検索対象となりません。

引用符やかっこによって検索要素をグループ化できます。

類似する文字に合致

ヘルプ ビューアの［検索］タブには［類似する文字に合致］オプションがあります。このオプションを選択すると、一般的な接尾辞を持つ単語をすべて検索できます。たとえば、このオプションを有効にして「run」を検索すると、「run」、「running」、「runner」が検索されます。「runtime」は検索されません。

正規表現

ヘルプ検索には、以下の正規表現を使用できます。

- * ：0 以上の文字の合致
- ? ：1 文字の合致
- 引用符内の文字列（"ab cd"）：逐語的に合致

検索対象	フィールドに入力する文字列
「analyze」、「analysis」、「analyses」、「analyzed」、「analyzing」を含むトピック	analy*
「analyzer」、「analyzed」を含むが「analyze」または「analyzers」は含まないトピック	analyze?
「analyze and generate」というフレーズを含むトピック	"analyze and generate"

演算子

ヘルプでの検索を絞り込むため、演算子（AND、OR、NOT、NEAR）を使用できます。検索文字列は左から右へ解釈されます。次の表に例を示します。

検索対象	フィールドに入力する文字列
「workspace」と「file」の両方を含むトピック	workspace AND file **または** workspace & file **または** workspace file
「workspace」と「file」のいずれかを含むトピック	workspace OR file **または** workspace | file
「workspace」を含むが「file」は含まないトピック	workspace NOT file **または** workspace | file
「workspace」の近くに「file」があるトピック（「workspace」と「file」の間は 8 文字以内）	workspace NEAR file
「workspace」を含むが「file」は含まないトピック、または「workspace」を含むが「directory」は含まないトピック	workspace NOT file OR directory

ネストされた表現

括弧を使用して表現をネストし、ヘルプで複雑な検索を行うことができます。かっこ内の表現が他の部分より先に解釈されます。表現のネスト可能なレベルは 5 レベルまでです。

検索対象	フィールドに入力する文字列
「workspace」を含むが「file」と「directory」のいずれかは含まないトピック	workspace NOT (file OR directory)
「workspace」を含み「file」と「project」が近くにあるトピック、または「workspace」を含み「directory」を含むが「project」が近くにあるトピック	workspace AND ((file OR directory) NEAR project)

UML モデリング

「UML モデリング」セクションの各章では、UML プロジェクトに固有の機能について説明しています。

2

モデルの操作

この章ではモデル ベースの開発について紹介します。ここでは、モデル バインディングの維持の方法について説明します。また、テキスト情報の構文カラー スキームについても触れます。

参照

第 i 章「ダイアグラムの操作」
第 i 章「UML 言語ガイド」

モデルとモデル要素

モデル ベースの開発

モデル ベースの特徴をもつ UML ツール セットは、複雑なモデルの作成と維持管理のための便利な機能を提供します。

操作方法は 2 あります。

- **ダイアグラム中心**の方法。モデルのダイアグラムの作成、編集をする際にモデルを作成します。
- **モデル中心**の方法。まず［モデル ビュー］ブラウザでモデルを作成してから、ダイアグラム ビューを定義します。

もちろん、2 つのパラダイムを組み合わせることもできます。

ダイアグラム中心のワークフロー

ダイアグラム中心のワークフローは、グラフィック言語を使用したことのあるユーザーにはよく知られた方法です。操作の進め方の例を以下に示します。

- ダイアグラムを作成する。
- ダイアグラムのエンティティを作成する。
- 定義したエンティティの詳細を記述するための、新しいダイアグラムを作成する。

この方法の利点は、新しいエンティティを作成すると、グラフィック コンテキストを使用できるようになるので、簡単かつ正確にモデルを作成できる点です。

モデル中心のワークフロー

モデル中心のワークフローは、モデル定義に存在するかどうかが確実ではないグラフィック表示に依存しません。ワークフローの例を以下に示します。

- モデル ブラウザでモデル要素を定義する。
- 新しいモデル要素をこのモデル構造内に配置する。
- モデルの関連パートを可視化する必要があるときに、ダイアグラムを作成する。
- モデル要素を［モデル ビュー］ブラウザからダイアグラムにドラッグすると、簡単にエンティティを可視化できる。
- モデル要素は、何度でも可視化できる。ダイアグラム ビューが異なる場合でも可能。

モデル ベース開発の結果、ダイアグラムにエンティティを記述するかどうかは、場合によって選択可能です。ダイアグラムのモデル表示の完全性を重視する場合、グラフィック表示されないエンティティをチェックするようにツールを設定できます。

モデル要素とプレゼンテーション要素

モデル要素

新しいオブジェクトまたは既存のオブジェクトに対して、新しい名前をつけて新しい定義を作成すると、ツールはモデルにオブジェクトが存在していないと認識します。これで新しいモデル要素が作成されます。このモデル要素はワークスペース ウィンドウの［モデル ビュー］に表示されます。

プレゼンテーション要素

ダイアグラムのシンボルは、モデル要素に基づいたプレゼンテーション要素です。通常、1 つのモデル要素に、プレゼンテーション要素をいくつでも配置できます。

要素プロパティ

クラス シンボルの属性名のように、あるプレゼンテーション要素のプロパティを変更すると、その変更はクラス内の他のプレゼンテーション要素にも反映されます。モデル要素が変更されると、この変更でプロパティに影響を受けたプレゼンテーション要素のすべてが更新されます。

(モデル ブラウザ、または、表示されたクラス シンボルのいずれかの）クラスに新しい属性を追加しても、この属性が、自動で、すべてのプレゼンテーション要素に表示されるわけではありません。もちろん、この属性をモデル内に配置して、プロパティを可視化するクラス シンボルに簡単に追加することもできます。

削除

クラス シンボルの属性を削除しても、シンボルの属性のプレゼンテーションが削除されるだけです。

モデルからの削除

［モデル ビュー］の属性を削除すると属性のモデル要素が削除され、これに伴い、その属性のすべてのプレゼンテーション要素も消失します（ダイアグラム内のプレゼンテーション要素を右クリックするかショートカット メニュー コマンドから［モデルの削除］を選択して削除することもできます)。

他のクラスで参照される属性タイプのように、他の場所から参照されるクラスを削除した場合、これらの参照のバインドが解除されます。

モデル要素

新しい要素の自動名前付け

モデル要素を定義する新しいシンボルを追加すると、現在のスコープで一意の名前となるようにシンボルにデフォルト名が作成されます。(テキストを選択せず）入力するだけで、デフォルトの名前を目的の名前に変更できます。

モデル要素のコピーと移動

モデル要素はコピー／貼り付けできます。

モデル要素を参照するシンボルをコピーしてこのシンボルを貼り付けると、既存のモデル要素のプレゼンテーションのみ新たに作成されます。これら 2 つのシンボルの一方を名前変更すると、他方のシンボルの名前も変更されます。2 つとも同じ要素のプレゼンテーションだからです。

ブラウザでモデル要素をコピーした場合、これを他の場所に貼り付けられます（ここでは、モデル要素自体のコピーとなります）。同じスコープに貼り付けると、同じ名前の定義が 2 つあることから競合が生じるため、チェッカーから何らかのレポートを受けます。どちらか一方のモデル要素の名前を変更すると、この競合は解消されます。

モデル要素がコピーできるのと同様に、定義もあるスコープから他のスコープへドラッグして移動できます。

テキストの強調表示

オブジェクトの場所

UML モデルの複数の場所で、たとえば［モデル ビュー］または出力ペインのオブジェクトをダブルクリックするか、出力ペインのオブジェクトに対して［検索］コマンドを選択して、定義を配置できます。上記の操作を実行すると、黄色のテキスト背景で強調表示された、正確なダイアグラムが表示されます。また、定義のプレゼンテーションが複数ある（あるいは、皆無の）可能性がある場合、プレゼンテーションの作成が起動します。

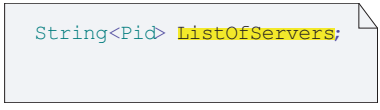

図 1: ロケーション マーカー

名前のナビゲーション

Ctrl キーを押しながらグレー表示されていない、下線のない名前をクリックすると、名前をナビゲートできます。これはツールチップにも表示されます。

プロパティ

必ずしも、モデル要素のプロパティのすべてを、ダイアグラムのプレゼンテーション要素から編集できるわけではありません。このため、モデル要素のプロパティを表示して、編集するためのプロパティ エディタが用意されています。プロパティ エディタはショートカット メニュー（要素を右クリックして、［プロパティ］にカーソルを合わせます）、または、Alt + Enter キーを押して開くことができます。

モデルのチェック

構文解析

ダイアグラム シンボル内のテキストを編集する際、テキストが正確かどうか解析され、モデルに追加されます。その後、このモデルをもとに、テキストはダイアグラム シンボルに書き戻されます。

このテキスト解析は、完全なモデル ベースのアプローチの結果です。（複数ある中から）1 つの特定の選択構文にテキストを書き込んだ場合、指定した書式が保存されないことがあります。

モデルの復元（F8）

構文解析により、変更箇所に構文エラーが見つかった場合、モデルを元の状態に復元できます。テキスト編集で構文エラーが選択された常態で、F8 キーを押して復元を実行します。この操作で編集したテキストがモデル情報から削除されます。

このコマンドを使用するときは以下の注意が必要です。コメントのような、モデルにバインドされていないテキストはすべて消去されます。モデルを初めて作成し、正確なモデル解析が行われていない場合、このコマンドですべてが消去されます。

名前のサポート

定義を参照したい場合、次のいくつかの方法を利用できます。

- プレゼンテーションの作成を使用して、完全なモデル間で素早くブラウズ、ナビゲートできます。
- **名前の完成**

  名前の最初の文字を入力してから（たとえば `ca`）、Ctrl + スペースバーを押すと、ツールが既存の名前にあてはめて、名前を完成しようとします（たとえば、`card`）。複数が合致する場合は、名前の完成スクロール メニューが開きます。特殊なケースをいくつか説明します。

  - ピリオド（「.」）の後の入力。名前の完成によって、左側の式の型に対してローカルまたは継承メンバー（構造的機能またはイベント クラス）である文字に一致する候補がリストされます。
  - スコープ修飾子（「::」）の後の入力。名前の完成によって、左側の式の名前空間にある候補がリストされます。
- **既存要素を参照**

  ダイアグラム要素作成ツールバーを使用して（モデル要素を定義または参照できる）新しいシンボルを作成し、ダイアグラム内でマウスの右ボタンをクリックすると、［既存要素を参照］というサブメニューのあるショートカット メニューが表示されます。このサブメニューにはシンボルの種類の定義がリストされ、ここから識別子を選択できます。

モデルの部分のチェック

［モデル ビュー］でチェックするモデルの部分を選択します。［選択部分のチェック］クイック ボタン（［分析］ツールバー）を使用します。

エラーと警告

チェッカーがモデルについて何らかの問題を検知すると、出力ウィンドウの［チェック］タブにレポートが表示されます。通常、ダイアグラムまたは、［モデル ビュー］ブラウザで、問題（警告とエラー）ごとに、その原因までさかのぼって追跡されます。この操作は、メッセージをダブルクリックするか、メッセージを選択して右クリックし、ショートカット メニューから［検索］を選択して実行します。

モデルとダイアグラム

ダイアグラム

モデルのさまざまなビュー

ダイアグラムは、モデルのある特定の側面やパートに焦点を合わせた、モデルのプレゼンテーションです。UML の機能の 1 つに、モデルのさまざまなビューを表示する機能があります。これは、モデル要素が複数の場所で参照されることを意味します。通常、モデルを維持する際にこれは問題となりますが、強力なモデル ベースのツールを活用することで、すべての参照が自動更新されるようになります。つまり、モデル要素のプロパティが変更されると、そのモデル要素を参照しているすべての場所でこの変更が反映されます。

プレゼンテーション要素

シンボル

シンボルは、モデル要素とは異なる**プレゼンテーション要素**です。シンボルが削除されても、モデル要素はモデル内に残ります。以下のいずれかの手順を実行すると、モデル要素が削除されます。

- ［モデル ビュー］ブラウザでモデル要素を削除する。
- シンボル上で［モデルからの削除］コマンドを実行する。

クラス シンボルで属性の名前を変更した場合、この変更は、クラス内の他のプレゼンテーションにも反映されます。

（モデル ブラウザ、または、表示されたクラス シンボルのいずれかの）クラスに新しい属性を追加しても、この属性が自動で、すべてのプレゼンテーション要素に表示されるわけではありません。もちろん、この属性をモデル内に配置して、プロパティを可視化するクラス シンボルに簡単に追加することもできます。

クラス シンボルの属性を削除しても、シンボルの属性のプレゼンテーションが削除されるだけです。［モデル ビュー］ブラウザで属性を削除すると、他のクラス シンボルの属性の表示もすべて消失します。

［モデル ビュー］ブラウザでクラス自体を削除すると、このクラスを参照するすべてのシンボルがダイアグラムから消失します。たとえば他のクラスでの属性タイプとして、他の場所でクラスが参照されている場合、クラスを削除するとこれらの参照のバインドが解除されます。

プロパティ エディタ

プロパティ エディタを開く

プロパティ エディタは、[モデル ビュー] またはダイアグラムの要素を選択し、ショートカット メニューから [プロパティ] を選択して開きます。プロパティ エディタは、ドッキング ウィンドウとして開きます。他のエディタと同じように、タイトル バーを右クリックして、ドッキングを解除したり他の場所にドッキングできます。プロパティ エディタは、ユーザーが閉じるまで開いたままとなります。

複数のウィンドウ

プロパティ エディタは複数開くことができます。これは、いくつかの要素のプロパティの設定を比較する場合などに便利です。複数のウィンドウを開けるようにするには、1 つのプロパティ エディタ ウィンドウで [選択部分をトラック] を無効にします。

プロパティ エディタ ウィンドウ

プロパティ エディタのビューは、上から順に以下の領域で構成されます（43 ページの図 2 を参照）。

- ウィンドウの左上。選択している要素の名前とアイコンが表示されます。
- [オプション] ボタン。現在のプロパティ エディタのオプションを設定するために使用します。
- フィルタ選択メニュー。表示する要素のプロパティを絞り込みます。
- [ステレオタイプ] ボタン。要素に適用するステレオタイプを選択するために使用します。このボタンをクリックして開くダイアログは、要素のショートカット メニューから [ステレオタイプ] メニュー項目を選択して開くダイアログと同じです。
- 要素のプロパティの表示と編集に使用するコントロール。この領域は、編集する要素と選択したフィルタによって動的に変化します。

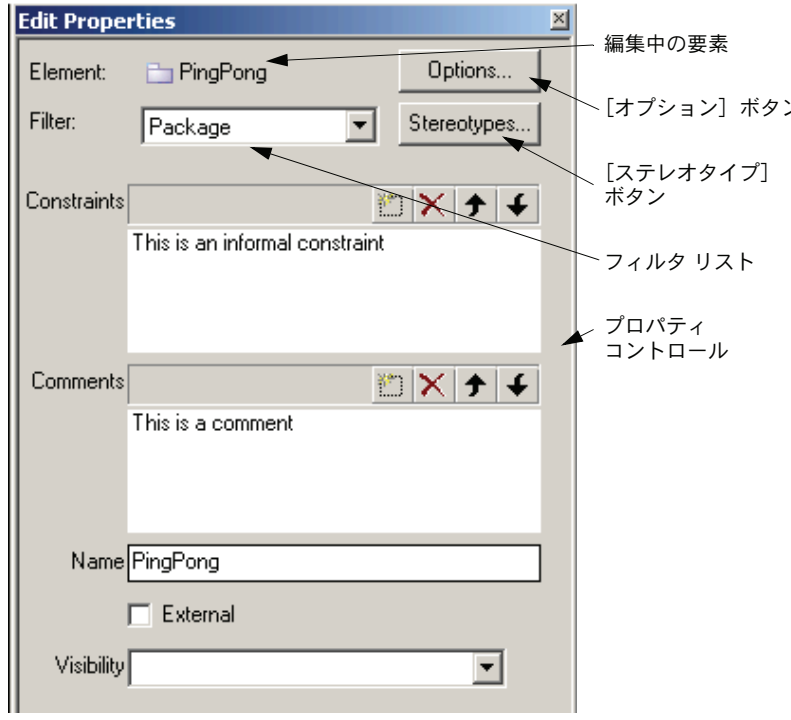

図 2: プロパティ エディタ

フィルタ リストには以下の項目があります（このとおりの順番とは限りません）。

- 編集中の要素のメタクラスの名前。
  この項目を選択すると、プロパティ エディタには要素のメタ特性値が表示されます（44 ページの「複数種類のプロパティ」を参照）。
- 編集中の要素に適用されている各ステレオタイプの名前。
  オプションの拡張（0..1）と必須拡張（1）を持つステレオタイプがメタクラスに適用されます。ステロタイプの拡張の詳細については、273 ページの「拡張性」を参照してください。ただし、非表示ステレオタイプ（<<hidden>> ステレオタイプが適用されたステレオタイプ）は表示されません。
- コメント
  この項目を選択すると、プロパティ エディタに編集中の要素に適用されたコメントが表示されます。コメントがない場合、要素にコメントを作成するボタンが表示されます。要素に複数のコメントがある場合、最初のコメントが表示されます。

すべてのプロパティ
この項目を選択すると、プロパティ エディタに編集中の要素のすべてのプロパティが表示されます。プロパティ コントロールの順序は、フィルタ リストの対応する項目順序と同じです。

インスタンスを選択している場合のプロパティエディ表示

インスタンスを選択している場合に、上で説明したプロパティエディタのコントロールの一部が意味を成さなくなります。その場合には該当コントロールは表示されません。

- ［Stereotypes］ボタンは表示されなくなります。インスタンスに対しては、ステレオタイプを適用できないからです。
- ［Options］ボタンは表示されなくなります。オプションの一部はインスタンスに当てはまらないからです。
- ［Filter］リストはインスタンスのシグニチャの情報に置き換えられます。

典型的な例としては、モデルビューであるインスタンス（たとえばステレオタイプインスタンス）を選択している場合にプロパティエディタの表示が上記のように修正されます。ただし、クラスのような構造型で型付けされた属性用にタグ付き値を編集している場合にも、インスタンスが選択されています。

複数種類のプロパティ

選択した要素には、原則的にメタ特性値とタグ付き値の 2 種類のプロパティを関連付けられます。どちらのプロパティもプロパティ エディタで編集できます。

メタ特性値

メタ特性値は、要素のメタクラスのメタ特性の値です。
要素の一連のメタ特性は固定されているので（また、UML 標準によってある程度限定されるので）新しいメタ特性を追加することはできません。既存の一連のメタ特性から、あるメタ特性の値だけを表示するように絞り込むことはできます。
このメタ特性値の例には、クラスの「Active」プロパティがあります。

タグ付き値

タグ付き値は、要素に適用されているステレオタイプの属性の値です。メタ特性値と違って、タグ付き値はかなり増える可能性があります。これは、要素に任意数のステレオタイプを適用でき、それぞれに任意数の属性を持たせることができるからです。たとえば、シンボルに表示するアイコンを指定する「Icon File」プロパティなどがあります。他の例を 50 ページの図 6 に示します。

プロパティ エディタのオプション

プロパティ エディタの［オプション］ボタン（45 ページの図 3 を参照）をクリックすると、［プロパティのオプションの編集］ダイアログが表示されます。このダイアログで設定するオプションは、現在のプロパティ エディタが開かれている間のみ有効です。つまり、2 つのプロパティ エディタを開いて、それぞれに別のオプションを設定できます。

注記

オプションの一部は、一般的な［オプション］ダイアログでも設定できます。そこで、すべてのプロパティ エディタに適用できるオプションを設定して保存できます。オプション値の一部はプロパティエディタの一般的なショートカット メニューでも変更できます。

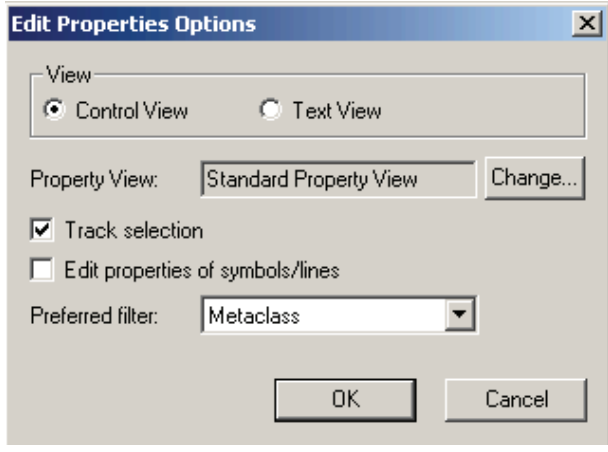

図 3: プロパティ エディタの［プロパティのオプションの編集］ダイアログ

View

プロパティ エディタでは、プロパティ値の編集にデフォルトで［コントロール ビュー］が使用されます。このビューには、要素プロパティの編集に使用するチェックボックスやプルダウン メニューなどがあります。タグ付き値については、UML 構文でのテキスト編集が可能です。テキスト編集は、［テキスト ビュー］で行うことができます。

［コントロール ビュー］で入力したテキスト フィールドの値は、フィールドのエディット モード終了時にモデルにコミットされます。

プロパティ ビュー

プロパティ エディタは、メタモデルを使用してカスタマイズできます。メタモデルによりメタクラスで使用できるメタ特性を指定すると、プロパティ エディタで要素に表示するプロパティを決定してこの情報できるようになります。メタモデルの使用方法の詳細については、52 ページの「プロパティ エディタのカスタマイズ」を参照してください。

選択部分をトラック

プロパティ エディタには、デフォルトで［モデル ビュー］またはダイアグラムで選択した要素のプロパティが表示されます。2 つの要素プロパティの比較を可能にするなど、選択部分のトラック機能を無効にしたほうがよい場合があります。このためには、設定を変更する要素のプロパティ エディタを開きます。プロパティ エディタの［オプション］ボタンをクリックし、［選択部分をトラック］の選択を解除します。これで、他の要素のプロパティ エディタを開き、この新しいプロパティ ウィンドウでモデル内の選択部分をトラックできます。

シンボル / ラインのプロパティを編集

シンボルまたはラインを選択している場合、プロパティ エディタにはデフォルトでシンボルまたはラインに対応するモデル要素のプロパティが表示されます。選択したシンボルまたはラインのプロパティを表示するよう設定するには、このオプションを選択します。

たとえば、クラス シンボルを選択している場合、通常は、プロパティエディタには対応するクラスのプロパティが表示されます。［シンボル / ラインのプロパティを編集］オプションを選択している場合は、対応するクラスではなくクラス シンボルのプロパティが表示されます。

使用するフィルタ

このオプションにより、新しい要素を選択した最優先されるフィルタを指定します。［メタクラス］、［ステレオタイプ］、［コメント］、［すべてのプロパティ］の 4 つの選択肢があります。次回編集項目を変更する際、このオプションが有効になります。

一般的なショートカット メニュー

プロパティ エディタには、コントロール以外の場所でマウスを右クリックしたときに表示される、ショートカット メニューがあります。47 ページの図 4 にショートカット メニューを示します。

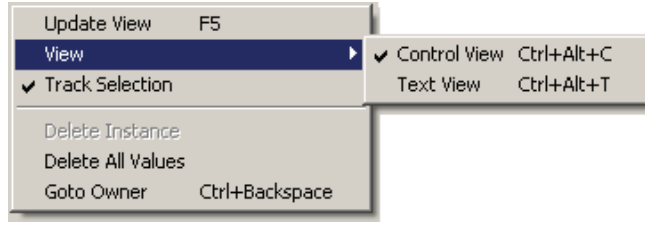

図 4: プロパティ エディタのショートカット メニュー

表示の更新

プロパティ エディタの表示を更新します。プロパティ エディタ以外で値が変更された場合など、プロパティ エディタの内容が自動更新されます。ただし、現在属性値が表示されているステレオタイプに属性が追加された場合や、プロパティ エディタを開いている間にアクティブ プロパティのメタモデルが変更された場合など、手動で表示の更新を行う必要があります。

表示

このメニュー項目は、[オプション] ダイアログ（45 ページの「プロパティ エディタのオプション」を参照）の対応するオプションのショートカットです。

選択部分をトラック

このメニュー項目は、[オプション] ダイアログ（45 ページの「プロパティ エディタのオプション」を参照）の対応するオプションのショートカットです。

インスタンスの削除

このメニュー項目は、適用された 1 つのステレオタイプのタグ付き値を編集する場合に使用できます。インスタンスに含まれるすべてのタグ付き値を削除し、ステレオタイプ インスタンス全体を削除できます。[ステレオタイプ] ダイアログを開き、適用ステレオタイプのリストから編集中のステレオタイプを削除するためのショートカットです。

すべての値の削除

このメニュー項目は、表示されているすべてのプロパティの値を削除するために使用します。デフォルトがあるプロパティはデフォルト値に戻され、ない場合は指定なしとなります。タグ付き値の編集の場合、このメニュー項目により、適用ステレオタイプ インスタンスを残し、すべてのタグ付き値を削除できます。

所有者へ移動

このメニュー項目は、編集中の要素の所有者のプロパティ ページへの移動に使用する便利なショートカットです。たとえば、クラス属性のプロパティを編集すると、［所有者へ移動］によって属性の所有者（クラスなど）を表示できます。

注記

編集のためにインスタンスを選択している場合は、プロパティエディタのショートカットメニュー項目の一部が使用できません。

コントロール ショートカット メニュー

各プロパティ コントロールのショートカット メニューも用意されています。このメニューの内容は、プロパティ コントロールの種類によって異なります。たとえば編集コントロールには、切り取り／コピー／貼り付けなどのメニュー項目があります。すべてのプロパティ コントロールに共通するメニュー項目を、48 ページの図 5 に示します。

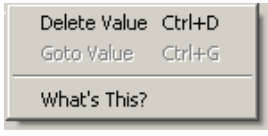

図 5: プロパティ コントロール、ショートカット メニューの例

値の削除

このメニュー項目は、プロパティ コントロールの値を削除するために使用します。プロパティにデフォルトがある場合はデフォルト値に戻ります。デフォルト値がない場合、指定値なしとなります。

値へ移動

リスト コントロールには、モデル内の他の要素のリストの値が表示されます。このコントロールでは、［値へ移動］メニュー項目を使用してリスト内で選択した要素に移動できます。

たとえば、ほとんどの要素には、編集中の要素に添付されたすべてのコメントを表示するコメント リストがあります。これらのコメントの 1 つを選択すると［値へ移動］メニュー項目が使用できるようになります。プロパティ エディタには選択したコメントのプロパティ（通常は 1 つのプロパティ、コメント テキスト）が表示されます。

これは何？

プロパティ コントロールに対応する属性（ステレオタイプまたはメタクラスの属性など）にコメントが添付されている場合、このメニュー項目が使用できます。このメニュー項目を選択すると、コメントにツール チップが表示されるようになります。ス

テレオタイプとメタモデルの設計者は、ステレオタイプとメタクラスのユーザーがプロパティ コントロールに入力すべき値がわかるように、ステレオタイプとメタクラスの属性にコメントを添付できます。

属性にコメントが添付されていない場合でも、「これは何？」テキストが表示されるコントロール（たとえばメタ特性値の表示など）もあります。これは、コントロールに入力する値がモデル要素に翻訳されるテキストである場合です。この場合は、コントロールに入力すべきテキストの種類がツールチップに表示されます。たとえば、コントロールに UML 式を入力する必要がある場合は、ツールチップに「式」と表示されます。

カラー コード

タグ付き値（メタ特性値以外の値など）の編集時、プロパティ エディタでタグ付き値の状態を示すためのカラー コード スキームを使用できます。

適用ステレオタイプ インスタンスで明示的に指定されたタグ付き値は、白いプロパティ コントロールで示されます。

ステレオタイプ インスタンスで指定されていないタグ付き値は、対応するステレオタイプ属性にデフォルト値がある場合、緑のプロパティ コントロールで示されます。

ステレオタイプ インスタンスで指定されていないタグ付き値は、対応するステレオタイプ属性にデフォルト値がない場合、黄色のプロパティ コントロールで示されます。

ステレオタイプの設計者はこのカラー コードを使用して、ステレオタイプ属性の意図をユーザーに知らせることができます。緑の値は、適切なデフォルト値があるので、値の指定は任意であることを表します。黄色の値は、その属性に適切なデフォルト値がないので、値の指定は必須であることを表します。

例 1: 色分けされた属性フィールドがあるステレオタイプ

3 つの属性を持つステレオタイプを考えます。50 ページの図 6 の例では、ステレオタイプ `MyStereo` がクラス x に適用されます。ユーザーが 2 つ目の属性に値を指定すると、このフィールドの色が黄色から白に変わります。

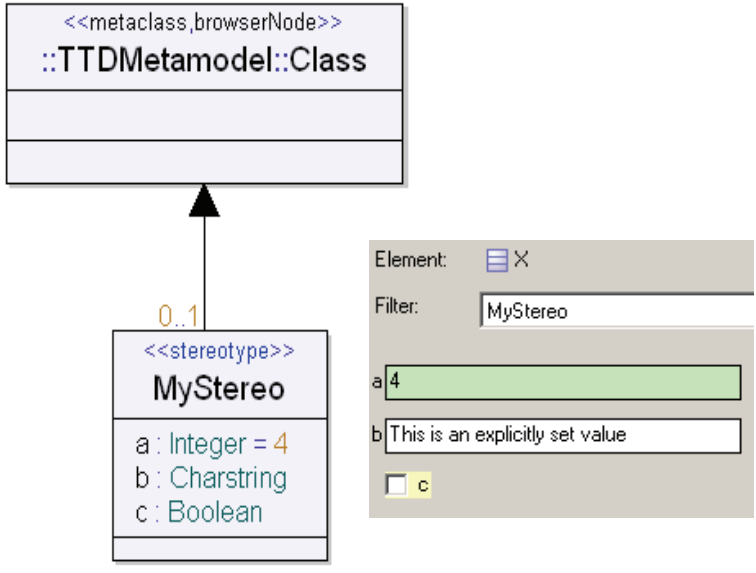

図 6 属性を持つステレオタイプ

あるコントロールのテキストに構文エラーがあるかどうかを示すための色分けもあります。構文チェックは、すべてのコントロールについて、そのテキストが U2 テキスト構文の文法にしたがっているかどうかを確認します。構文エラーのあるテキストは赤で示され、構文エラーがないテキストは黒で示されます。テキストが赤で示された状態のまま編集を終了すると、メタ特性値は編集前の正しい値に戻ります。このようなカラー コードによって、編集中に誤って正しい情報を失うのを防ぐとこができます。

例 2: プロパティ エディタでの構文エラーの色分け

ポートの 'Realizes' メタ特性は ID リストを要求します。'signal' は UML のキーワードなので、メタ特性の現在のテキスト（51 ページの図 7 参照）には構文エラーがあります。したがって、この状態で編集モードを終了すると、値は編集前の正しい値（いかなる値でも）に戻ります。

図 7: メタ特性値のエラーと正しい値

プロパティ エディタのカスタマイズ

要素に適用するステレオタイプを設計する際、2 つのユーザー ロールが考えられます。それは、ステレオタイプの属性を決定するステレオタイプの設計者と、ステレオタイプを要素に適用してステレオタイプ属性にタグ付き値を指定する、ステレオタイプのユーザーです。同一人物がこれら両方のロールを持つ場合もありますが、通常はステレオタイプの設計者とユーザーは別人です。

このセクションでは、設計者ロールに焦点を当て、新しいステレオタイプあるいはメタクラスの設計方法について説明します。また、設計者が望む方法でステレオタイプやメタクラスのインスタンスを編集できるように、プロパティ エディタをカスタマイズする方法についても説明します。

プロパティ エディタでは、カスタマイズのために通常 TTDExtensionManagement プロファイルというプロファイルを使用します。これはモデルのライブラリ フォルダにあります。

ステレオタイプの設計

プロパティ エディタで使用するステレオタイプの設計は、以下の手順によって行います。

- 新しいステレオタイプの定義を保管する場所を決定します。ステレオタイプをローカルの現行プロジェクトでのみ使用する場合、適用する要素と同じファイルに追加します。通常は、複数プロジェクトでステレオタイプを使用する場合が多いので、独自のファイルに保存されるパッケージに保管します。ステレオタイプを持つ再利用可能なパッケージを、プロファイル パッケージといいます。そのようなパッケージをツールのライブラリにロードする方法については、1696 ページの「アドイン」を参照してください。
- ステレオタイプに適切な名前を付けます。ステレオタイプの名前は、[ステレオタイプ] ダイアログ、プロパティ エディタのフィルタ リスト、ダイアグラムのシンボルなどに表示されます。
  `TTDExtensionManagement::instancePresentation` ステレオタイプを使用して、ステレオタイプによりわかりやすい表示名を付けると便利な場合があります。指定した表示名は、[ステレオタイプ] ダイアログとプロパティ エディタのフィルタ リストに表示されます。詳細については、55 ページの「TTDExtensionManagement プロファイル」を参照してください。
- ステレオタイプにコメントを付けます。コメントは、ステレオタイプの目的、適用できる要素に関する制約などを表します。コメントは、ステレオタイプを選択した際、[ステレオタイプ] ダイアログの下部に表示されます。また、ステレオタイプにツールチップとして表示されます。
- ステレオタイプに適切なタイプと多重度を持つ属性を追加します。ステレオタイプの属性はどのようなタイプと多重度でもかまいませんが、プロパティ エディタで [コントロール ビュー] を使用して編集する際にサポートされる一連のタイプと多重度を考慮する必要があります。サポートされないタイプまたは多重度のある属性を使用していると、[コントロール ビュー] で属性を変更できません。この場合、[テキスト ビュー] で編集する必要があります。

以下の表に、サポートされるタイプと多重度の組み合わせ、またそれぞれに使用されるグラフィカル コントロールを示します。メタクラスの属性のみに適用可能なタイプと多重度の組み合わせについては、54 ページの「メタクラスの設計」の表を参照してください。

属性のタイプと多重度	プロパティ コントロール
Boolean 単一の多重度	CheckBox
Charstring 単一の多重度	EditControl
Charstring 複数の多重度	EditList
Integer, Natural, Real 単一の多重度	EditControl
Enumeration 単一の多重度	DropDownMenu （各リテラルに 1 項目）
Enumeration 複数の多重度	CheckBoxList （各リテラルに 1 チェックボックス）
Structured type（クラスなど） 必須、単一の多重度（1）	Group （構造型の各属性に 1 つのサブ コントロール）
メタクラス タイプ 単一の多重度 参照	DropDownMenu （メタクラスの可視定義ごとに 1 項目）
メタクラス タイプ 複数の多重度 参照	EditControl

上記タイプのシンタイプもサポートされます。

- 属性のデフォルト コントロールが適切ではない場合、属性に `TTDExtensionManagement::extensionPresentation` ステレオタイプを適用し、タグ付き値としてカスタム コントロールを指定できます。詳細については、55 ページの「TTDExtensionManagement プロファイル」を参照してください。
- ステレオタイプのプロパティ ページに「値以外」のコントロールを追加できます。たとえば、プロパティ ページに静的テキストまたはボタンを追加できます。このためには、ステレオタイプに `TTDExtensionManagement::instancePresentation` ステレオタイプを適用し、`nonValueControls` 属性のタグ付き値として追加のコントロールを指定します。

- 各ステレオタイプ属性にコメントを添付できます。コメント テキストは、属性に対応するコントロールの［これは何？］ショートカット メニュー項目を選択すると表示されます。
- ステレオタイプ間で継承を利用できます。派生したステレオタイプのプロパティ ページには、派生ステレオタイプの属性の後に、ベースのステレオタイプ属性が表示されます。
- ステレオタイプ適用可能な要素の種類を指定します。このためには、ステレオタイプとメタクラスの間に拡張を設定します。これは、指定したメタクラスにステレオタイプを適用できることを示します。UML セマンティックでは、ステレオタイプに拡張がない場合、いかなる要素にも適用できません。
  拡張メタクラスのすべてのインスタンスにステレオタイプを自動的に使用可能とする場合、拡張を必須に設定（拡張ラインに「1」と入力）します。これで、編集中の要素に適用しなくても、プロパティ エディタのフィルタ リストにこのステレオタイプが表示されるようになります。
  ステレオタイプを手動で適用する場合、拡張を任意に設定（拡張ラインに「0..1」と入力）します。
  複数の拡張を使用することもできます。指定したメタクラスのどの要素にも、ステレオタイプを適用できます。

これで、新しいステレオタイプをテストする準備が整いました。適切な種類の要素（ステレオタイプによって拡張されたメタクラスの要素など）を作成します。作成した要素の場所からステレオタイプが見えるようにしてください。作成した要素のプロパティ エディタを開き、新しいステレオタイプのプロパティ ページを確認します。拡張を任意と指定した場合、［ステレオタイプ］ボタンを使用してまずステレオタイプを適用します。

メタクラスの設計

メタクラスの設計スコープは、ステレオタイプの設計の場合とほとんど同じです。大きな違いは、プロパティ エディタでメタクラスを使用できる要素の指定方法です。メタクラスでは、これはメタクラスを記述するクラスに <<metaclass>> ステレオタイプを適用することで実行されます。通常のクラスではなくメタクラスにするのが、このステップです。base 属性のタグ付き値によって、新しいメタクラスのベースとする組み込み UML メタクラスの名前を指定します。

注記

メタクラスの設計方法を覚える手始めとして、すべてのモデルでライブラリとして使用可能な TTDMetamodel プロファイルを学習すると良いでしょう。次のセクションで、独自のメタクラスとその属性のベースとして使用する、内蔵メタクラスとメタ特性の名前に関する情報を示します。また、指定メタクラスに対応するようプロパティ エディタをカスタマイズするための TTDExtensionManagement プロファイルについても説明します。
これは、プロパティ エディタの［オプション］ダイアログで［Standard Property View］と呼ばれる、TTDMetamodel です。

ステレオタイプとは異なり、メタクラスにはまったく新しい属性は指定できません。メタクラスのすべての属性は、基底メタクラスの既存メタ特性をベースにする必要があります。このためには、メタクラス属性に metafeature ステレオタイプを適用します。メタクラス属性の名前が対応するメタ特性と同じ名前の場合、タグ付き値 base は省略できます。その他の場合、指定は必須です。

注記

注意深いユーザーは、TTDMetamodel 内のメタモデル属性の一部が基底メタクラスのメタ特性に対応していないことに気づくでしょう。クエリ機能があり、<<queryFeature>> ステレオタイプを使用して、モデルからエンティティを算出するクエリエージェントを指定します。クエリ機能はプロパティエディタには表示されません。モデルビューにのみ表示されます。

以下の表に、メタクラスの属性にのみ適用可能なサポートされるタイプと多重度の組み合わせ、またそれぞれに使用されるグラフィカル コントロールを示します。ステレオタイプ属性に有効な組み合わせを示す表（ステレオタイプの設計）と比較してみてください。

属性のタイプと多重度	プロパティ コントロール
Metaclass type 単一の多重度 合成	EditControl
Metaclass type 複数の多重度 合成	EntityList

新しいメタクラスが設定できたら、パッケージに配置し、そのパッケージを独自のファイルに保管します。定義済みステレオタイプ <<propertyModel>> をパッケージに適用しておく必要があります。そして、プロファイルをロードするためのアドインの通常の作成手順を行います。プロファイルをロードしたら、プロパティ エディタの［オプション］ボタンを使用して、プロパティ エディタで使用するプロパティ ビューとしてプロファイル パッケージを指定します。

TTDExtensionManagement プロファイル

TTDExtensionManagement プロファイルには、独自のステレオタイプとメタクラスのプロパティ ページのカスタマイズを可能にする、ステレオタイプとクラスが含まれています。ここでは、このプロファイルの詳細について、その使用例を挙げて説明します。

ステレオタイプ

プロファイルには、プロパティ エディタと関係のある 3 つのステレオタイプ instancePresentation、extensionPresentation、filterStereotypes が含まれています。

instancePresentation

instancePresentation ステレオタイプは、ステレオタイプまたはメタクラスに適用して、ステレオタイプまたはメタクラスのインスタンスのプロパティ エディタでの表示方法をカスタマイズできます。

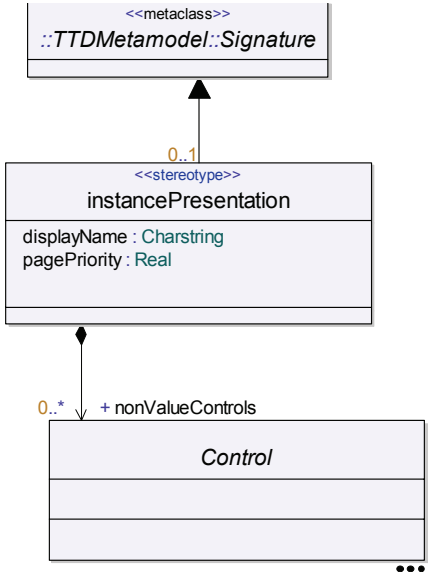

図 8: <<instancePresentation>> ステレオタイプ

displayName: Charstring

この属性は、ステレオタイプまたはメタクラスのインスタンスの表示名を指定します。表示名は、プロパティ エディタのフィルタ リストと［ステレオタイプ］ダイアログに表示されます。また、ツールの他の場所、たとえばツールチップや［モデル ビュー］などにも表示されます。

この属性にタグ付き値を何も指定しないと、表示名としてステレオタイプまたはメタクラスの名前が使用されます。

pagePriority:Real

この属性は、プロパティ エディタのフィルタ リストとプロパティ ページ（フィルタを使用する場合）での表示順序を制御します。ページ優先順位が高いステレオタイプは、低いページ優先順位番号を持つステレオタイプのインスタンスより前に配置されます。どのようなページ優先順位を指定しても、指定なしより優先順位が高いと見なされます。

注記

ページ優先順を指定する場合は、単一の数値を使用してください。それ以上複雑な式は評価されません。

nonValueControls:Control[*]

この属性は、特定の属性に対応しないプロパティ ページ内のコントロールなど、「値以外」のコントロールを指定します。例として静的テキストなどの「付属品」が挙げられますが、ボタンなどのように振る舞いを伴うコントロールの場合もあります。

extensionPresentation

`extensionPresentation` ステレオタイプ（57 ページの図 9）は、ステレオタイプまたはメタクラスの属性に提供し、属性に対応するコントロールのプロパティ エディタでの表示をカスタマイズできます。

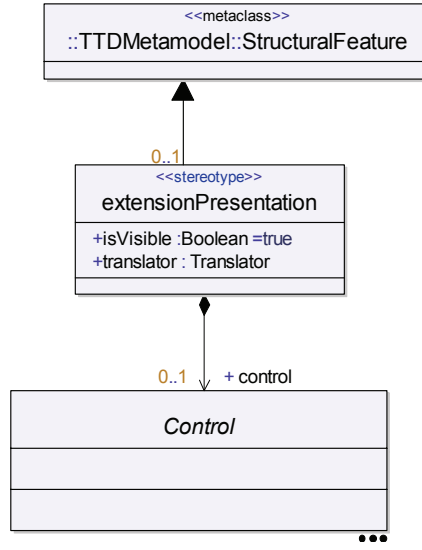

図 9: <<extensionPresentation>> ステレオタイプ

isVisible:Boolean

この属性は、属性のコントロールのプロパティ エディタでの表示／非表示を制御します。属性のコントロールを完全に非表示にする場合は、値を偽に設定します。

translator: Translator

この属性は、メタクラスのタイプを持つパートに排他的に使用します。54 ページの「メタクラスの設計」で説明したように、このような属性には、プロパティ エディタでは EditControl（単一の多重度の場合）または EntityList（複数の多重度）が使用されます。この場合、コントロールに入力されたテキストは UML テキスト構文なので、テキストの解釈のためトランスレータが必要です。Translator 列挙には、UML 文法の使用可能なエントリ ポイントごとに 1 つのリテラルが含まれます。Translator 列挙は非表示（内部）プロファイルにありますが、そのリテラルの名前は次のようにして表示できます。

- 右クリックして［既存要素を参照］を選択し、クラス図で列挙シンボルを作成する。
- 表示されたリストから、`U2ParserProfile::Translator` を選択する。
- 列挙シンボルを右クリックし、［表示／非表示］サブメニューから［Show Literals］を選択する。

注記

［参照の一覧表示］コマンド（ショートカット メニュー）を使用して、TTDMetamodel プロファイルでの Translator 列挙の使用方法を確認できます。たとえば、リテラル PEP_Multiplicity の参照のリストから、StructuralFeature::Multiplicity 属性のトランスレータとして使用されていることがわかります。このように、このトランスレータは、UML の多重度構文を解析するために使用されます。

control:Control[0..1]

52 ページの「ステレオタイプの設計」で説明したように、プロパティ エディタは属性のタイプと多重度、ときには集約の種類に応じて、デフォルトのコントロールを使用します。`control` 属性は、属性のデフォルト以外のコントロールの使用、あるいはデフォルト コントロールのプロパティの変更を可能にします。

例 3: ［テキスト ビュー］を使用したカスタム コントロールの指定

```
extensionPresentation(.
  control = EditControl(.
    text = "My Control",
    autoLayout = GrowRight
  .)
.)
```

Control クラスは抽象クラスです。このクラスはプロパティ エディタでサポートされるコントロールごとに 1 つの派生クラスを持ちます。

filterStereotypes

`filterStereotypes` ステレオタイプは、パッケージに適用して、そのパッケージ内の要素を選択したときにプロパティ エディタに表示されるステレオタイプの数を減らせます。

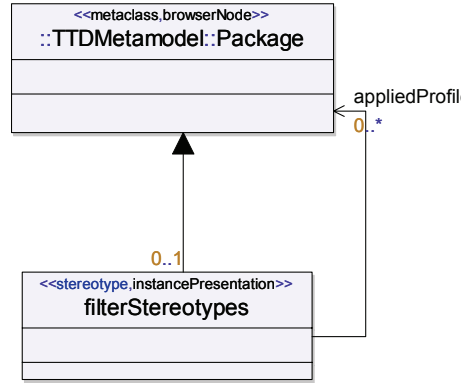

図 10: <<filterStereotypes>> ステレオタイプ

appliedProfile:Package[*]

このリストのプロファイル パッケージを指定すると、プロパティ エディタと［ステレオタイプ］ダイアログには、filterStereotypes ステレオタイプが適用されたパッケージで選択した要素に対して、これらのパッケージで定義されたステレオタイプのみ表示されます。

コントロール モデル

TTDExtensionManagement プロファイルには、プロパティ エディタで使用されるグラフィカル コントロールを表すさまざまなコントロール クラスがあります。利用可能なすべてのクラスを確認するには、クラス図 Controls を参照してください。

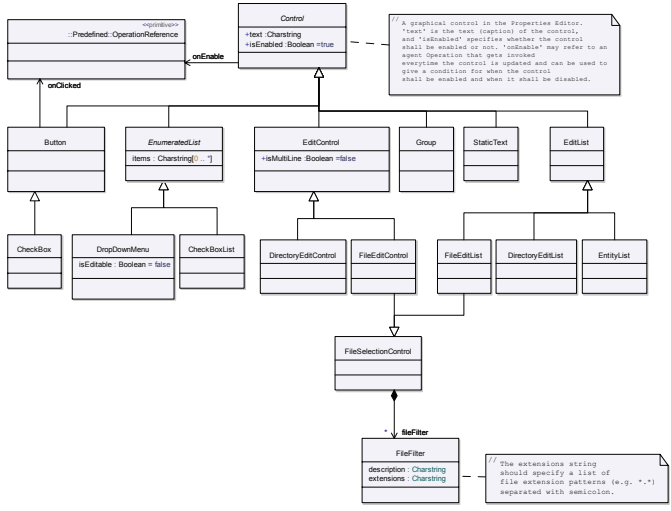

図 11: コントロール クラス

Control

Control クラスは、すべてのコントロール クラスの基底クラスです。

text: Charstring

この属性は、コントロールに使用するキャプションを指定します。この属性に何も指定しないと、キャプションは編集中のステレオタイプまたはメタクラスの属性の名前になります。

isEnabled:Boolean

デフォルトで、コントロールは有効になっています。つまり、表示された値の編集に使用できます。この属性を無効に設定すると、コントロールは無効になります。状況によっては、この属性の設定を無視して、プロパティ エディタによってコントロールを無効にすることがあります。編集中の要素に含まれるファイルが読み取り専用の場合、また、派生したメタクラス属性の場合などがこれに当てはまります。

onEnable:Operation

この属性は、コントロールを有効にすべき時期を動的に制御します。この属性にエージェント操作が指定された場合、プロパティ エディタによってコントロールを有効にするかどうかを決定する際、必ず呼び出されます。エージェント呼び出しのモデル コンテキストは編集中の要素です。呼び出しには以下のパラメータがあります。

- `[out] enable :Boolean`

  コントロールを無効にするには、エージェントによってこの out パラメータを無効にします。デフォルトで、コントロールは有効になっています。
- `stereotypeInstance :Entity`

  プロパティ ページで編集中の、コントロールを含むステレオタイプ インスタンス。このパラメータは、編集中のインスタンスがステレオタイプ インスタンスの場合のみ渡されます。

注記

`isEnabled` を無効にすると、`onEnable` エージェントは起動しません。

参照

第 57 章「エージェント」

Button

`Button` クラスは、クリック可能なボタンを表します。これは、値の編集用ではなく、プロパティ ページの値以外のコントロールに使用されます。CheckBox はトグル ボックス コントロールのある特殊なボタンで、ブール値の編集に使用できます。

onClicked:Operation

この属性は、ボタンをクリックしたときに実行される振る舞いを指定します。この属性により、ボタンをクリックしたときに実行されるエージェント操作を指定できます。エージェント呼び出しのモデル コンテキストは編集中の要素です。エージェント呼び出しにパラメータはありません。

EditControl

`EditControl` は、文字列値の編集に使用できます。編集中の文字列がディレクトリまたはファイルの名前の場合、特化された 2 つのバージョンのクラスが使用できます。これらのクラスにより、ディレクトリまたはファイルの選択ダイアログを開く参照ボタン［...］を追加できます。これで、コントロールに名前を手動で入力する必要がなくなります。

InstanceEditControl と呼ばれる特殊な EditControl があります。このコントロールはインスタンス（クラスのインスタンスなど）の編集のために使用されます。インスタンスはテキスト構文を使用してコントロール内に表示されますが、編集のためには参照ボタン［...］を使用します。このボタンを押すと、もう 1 つのプロパティエディタが開き、選択したインスタンスを編集できます。

isMultiLine:Boolean

デフォルトで、編集コントロールにはテキストが 1 行だけ表示されます。この属性を有効にすると、コントロールで複数行の編集が可能になります。同時に 2 行以上のテキストを表示するときは、コントロールの上下サイズを拡大する必要があります。この方法については、PositionedControl を参照してください。

EditList

EditList コントロールは、文字列のリストの編集に使用できます。このコントロールには、リストに新しい文字列を作成するボタンとリストから選択した文字列を削除するボタンが含まれます。さらに、選択した文字列をリスト内で上下に移動するための 2 つのボタンもあります。文字列の移動には、直接文字列を選択してドラッグアンドドロップする方法もあります。

図 12: 文字列の作成、削除、移動用のボタンをもつ EditList コントロール

編集中の文字列がディレクトリまたはファイルの名前の場合、EditList の特化バージョンである 2 つのクラスを使用できます。このクラスは、DirectoryEditList と FileEditList であり、ディレクトリオープン用またはファイル選択用のダイアログを開く参照ボタン [...] を追加します。これによって直接名前を入力する手間を省けます。

また、EntityList と呼ばれる特別な EditList があります。これは、他のメタクラスで型付けされる合成であるメタクラス属性（メタ特性など）のコントロールとして使用できます。EntityList 内の各編集項目は、モデル内の要素です。このような要素についてコントロールに表示される文字列は、その要素の UML テキスト構文です。

もう 1 つの特別な EditList は InstanceEditList です。これは、インスタンス（クラスのインスタンスなど）のリストを編集するために使用されます。インスタンスはテキスト構文を使用してコントロール内に表示されます（各行に 1 インスタンス）。インスタンスを編集するには、そのインスタンスをダブルクリックし、表示される参照ボタン [...] をクリックします。この操作でもう 1 つのプロパティエディタが開き選択したインスタンスの編集ができます。

StaticText

StaticText は、プロパティ ページの付属品として使用できる、値以外のコントロールです。これは、プロパティ エディタのコントロールに対する値の指定方法を示す静的テキストを追加する場合などに使用できます。

EnumeratedList

EnumeratedList は、列挙型要素のリストを編集するコントロールの共通ベースとなる抽象クラスです。このクラスの 2 つの具体的な特化として、DropDownMenu と CheckBoxList があります。

items: Charstring[*]

列挙型リストを列挙タイプの属性のコントロールとして使用される場合、列挙のリテラルごとに 1 項目が含まれます。各項目の名前は、デフォルトで対応するリテラルの名前です。`items` 属性の値として文字列のリストを指定すると、リストの項目名をカスタマイズできます。

DropDownMenu

`DropDownMenu` は、ドロップダウンメニューで編集を行う項目のリストです。

isEnabled:Boolean

デフォルトで、ドロップダウン メニューにある項目は 1 つしか選択できません。この属性を有効にすると、ドロップダウン メニューの編集が可能になり、項目名の手入力ができるようになります。

CheckBoxList

`CheckBoxList` は、チェックボックスのリストで編集を行う項目のリストです。したがって、このコントロールは、複数のリスト項目の選択が可能です。

Group

`Group` は、他のコントロールを包含するコンテナ コントロールです。通常は、構造型で多重度 1 のパート型属性用のコントロールとして使用します。構造型の属性ごとに 1 つのサブコントロールが含まれます。

ColorControl

`ColorControl` 整数型の属性に使用できます。値は、RGB の 3 つのコンポーネントで現される色参照として解釈されます。

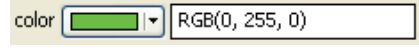

図 13: 緑色を値として指定した ColorControl

色値は矢印ボタンをクリックして表示される標準の色設定リストボックスか、RGB に対応する数値の直接入力で編集できます。

QueryControl

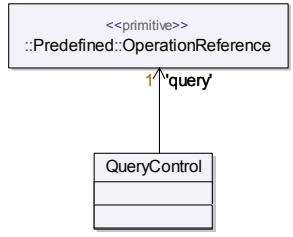

図 14: QueryControl の定義

QueryControl の外観は DropDownMenu と似ています。ただし、エンティティ数の固定されたリストではなく、クエリを通して動的にエンティティ数が変動するリストになっています。コントロールの値はクエリの結果から選択されたエンティティへの参照です。

query: Operation

この属性は、リストにデータを入れるために実行されるクエリエージェントへの参照です。

NavigationButton

NavigationButton はメタクラスによって型付けされる、単一多重度のメタ特性向けのコントロールとして使用できます。つまり、コントロールの値はモデル内の他のエンティティへの参照です。ボタンが押されると、そのエンティティを表示するプロパティページが開きます。

Navigation ボタンはモデル内の 2 つのエンティティに関係性があるときに使用でき、一方のエンティティのプロパティページからもう一方のページへのナビゲーションを容易にします。

GotoOwnerButton

GotoOwnerButton は、編集している要素の合成の親要素にナビゲートするための NavigationButton の特化形です。

ValueControl

コントロール クラスには、値の表示と編集が可能な ValueControl クラスを継承するものがあります。

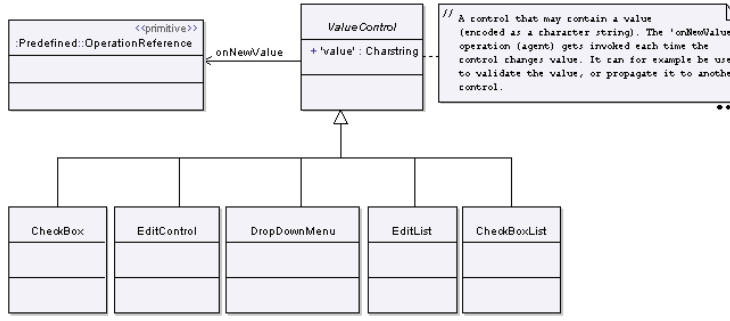

図 15: ValueControl クラス

value: Charstring

この属性は、プロパティ エディタの内部でコントロールの値の表現を保持するために使用されます。しかし、これを明示的に使用してコントロールに常に特定の値を表示させることもできます。

onNewValue:Operation

この属性は、コントロールに新しい値が入力されたときに実行されるエージェント操作を指定します。これは、コントロールに入力された値の有効性の確認や、他のコントロールへの値の移動のために使用できます。新しい値が設定される直前に呼び出されます。編集中の要素をモデル コンテキストとし、以下のパラメータを持ちます。

- `attribute :Entity`

  編集中の属性（ステレオタイプまたはメタクラス属性）。
- `newValue :Entity`

  コントロールに設定される新しい値。
- `stereotypeInstance :Entity`

  編集中の属性がステレオタイプ属性の場合、このパラメータは変更しようとするステレオタイプ インスタンスです。それ以外の場合、このパラメータは渡されません。

PositionedControl

`PositionedControl` クラスは、図形としての位置とサイズに関連するコントロールのプロパティを表します。デフォルトで、コントロールの配置を決定するため、プロパティ エディタは単純なオートレイアウトを適用します。属性は左揃えで上から下に

配置されます。コントロールのオートレイアウト ポジションは前のコントロールとの関連で計算されます。PositionedControl クラスの属性により、このレイアウトをある程度カスタマイズできます。

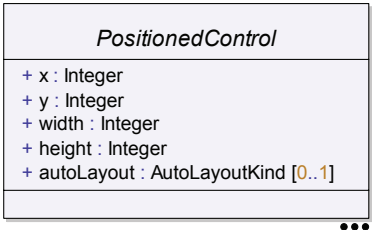

図 16: PositionedControl

x:Integer

この属性に値を指定して、コントロールのデフォルトの水平ポジションを置き換えます。

y:Integer

この属性に値を指定して、コントロールのデフォルトの垂直ポジションを置き換えます。

注記

コントロールのデフォルト位置を上書きするには、x 軸と y 軸の両方の値を指定する必要があります。コントロールに指定した位置は、デフォルトを使用する以降のコントロールにも影響を及ぼします。

width:Integer

この属性に値を指定して、コントロールのデフォルトの幅を置き換えます。

height:Integer

この属性に値を指定して、コントロールのデフォルトの高さを置き換えます。

autoLayout: AutoLayoutKind

この属性は、プロパティ エディタ ウィンドウのサイズ変更がコントロールに与える影響を決定する、オートレイアウト アルゴリズムのオプションを指定します。この属性には、以下の値を使用できます。

- `GrowRight`

  プロパティ エディタ ウィンドウのサイズを拡大すると、コントロールが右方向に拡大します。この振舞い方はほとんどのコントロールのデフォルトです。

- `GrowBottom`

  プロパティ エディタ ウィンドウのサイズを拡大すると、コントロールが下向に拡大します。
- `GrowRightAndBottom`

  プロパティ エディタ ウィンドウのサイズを拡大すると、コントロールが右下方向に拡大します。

プレゼンテーションの作成

［プレゼンテーションの作成］ダイアログは、［モデル ビュー］から［新規］コマンドを使用してダイアグラムの作成を行う代わりに、モデルへの適切なエントリ ポイントを提供します。このダイアログは、要素のショートカット メニューから開くことができます。

［プレゼンテーションの作成］ダイアログ

［プレゼンテーションの作成］ダイアログにはタイトルと一連のタブが表示されます。ダイアログのタイトルには、［プレゼンテーションの作成］の対象となる現在のエンティティのタイプと名前が表示されます。個々のタブには、タブの説明と選択肢が表示されます。

タブの選択肢をクリックすると、ダイアログが閉じ、必要に応じてモデル要素、シンボル、ラインまたはダイアグラムが作成され、それらにナビゲートします。

新しいシンボル

［新しいシンボル］タブを使用して、既存ダイアグラム内で現在のエンティティのシンボルを作成するか、またはエンティティのプレゼンテーション要素を含む新規ダイアグラムを作成して、シンボルを作成できます。

ダイアグラムの作成

［ダイアグラムの作成］タブでは、モデル ビューの作成ルールに従って新しいダイアグラムを作成します。このタブで、現在のエンティティの下にダイアグラムを作成できます。これは、［モデル ビュー］のショートカット メニューから［新規］を選択して、ダイアグラムを作成するのと同じです。

［場所］カラム

モデル内の選択肢の場所です。

［ダイアグラム名］カラム

選択肢の名前です。

［アイテムの種類］カラム、［ダイアグラムの種類］カラム

記述されたエンティティの種類です。たとえば、クラス、シンボル、クラス図などがあります。

参照

モデルのナビゲートと作成

第 i 章「ダイアグラムの操作」の 106 ページ、「シンボルの追加」

モデル ナビゲータ

モデル ナビゲータは、出力ウィンドウの［ナビゲート］というタブのことです。このタブによって、モデルのエンティティのあらゆる側面をブラウズ、またはナビゲートできます。

モデル ナビゲータは、モデルのナビゲーションに適切かつ強力なツールを提供します。［モデル ビュー］には階層構造のスコープ ビューでモデルが表示されますが、モデル ナビゲータにはさまざまなビューがあり、モデルの内部関係に基づいてモデルを詳しく調べられます。

また、モデル ナビゲータにより、以下を実行できます。

- ダイアグラムを選択して表示する。
- 現在のエンティティを示すシンボルまたはラインへナビゲートする。
- 現在のエンティティに関連したエンティティへのナビゲーション ショートカットをとる。

［モデル ビュー］のショートカット メニューまたはエディタのショートカット メニューから［モデル ナビゲータ］を選択すると、モデル ナビゲータが開きます。

モデルのナビゲートと作成

ダイアグラムまたはその要素をダブルクリックすると、モデルのナビゲートまたは作成が可能になります。

- ダブルクリックした要素を表すプレゼンテーション要素がある場合、この要素のダイアグラムの［ナビゲート］タブが表示される。
- ダブルクリックした要素を表すプレゼンテーション要素がない場合、［プレゼンテーションの作成］ダイアログが表示される。

モデル ナビゲータのタブ

［モデル ナビゲータ］タブ自体にも一連のタブがあります。これらのタブには、タブの説明と選択肢が表示されます。表示されるタブは現在のエンティティによって異なります。ウィンドウが表示されたときに選択されているタブは、以下の基準に従って決まります。

- 最後に使用したタブ
- 適切なタブで最優先されるもの

各カラム ヘッダーの右側の垂直バーをドラッグして、カラム幅を変更できます。

ソート

タブの選択肢は、名前カラムを基準にして昇順にソートされています。タブに名前カラムがない場合、タイプまたはインデックス番号カラムが基準になります。

カラム ヘッダーをクリックして手動でソートすることもできます。もう一度クリックするとソートの順番が逆になります。

タブのカテゴリ

モデル ナビゲータのタブは、以下の 2 つのグループに分類できます。

- ［モデル ビュー］またはダイアグラムに選択肢が表示されるタイプのタブ。このグループには**プレゼンテーション タブ**と**リンク タブ**があります。
- モデル ナビゲータのフォーカスを新しいモデル要素に移動する（CTRL+ マウスクリック）タイプのタブ。これらのタブは**エンティティ タブ**と呼ばれます。

それぞれのタブ グループの詳細を以下に示します。

- **プレゼンテーション タブ**

  プレゼンテーション タブの選択肢をクリックすると、ダイアグラムのシンボルやライン（［シンボル］タブ）、または、ダイアグラム自体（［ダイアグラム］タブ）にナビゲートします。
- **リンク タブ**

  ［リンク］タブの選択肢をクリックすると、ダイアログが閉じて、他のリンクの終端にナビゲートします。
- **エンティティ タブ**

  エンティティ タブの選択肢を CTRL+ クリックすると、クリックした選択肢がモデル ナビゲータにフォーカスが移ります。つまり、クリックした選択肢が現在のエンティティとなります。現在のエンティティは［モデル ビュー］で選択します（可能な場合）。このカテゴリには、［パッケージ］、［特性］、［お気に入り］、［定義］、［ショートカット］、［参照］、［モデル インデックス］、および［最近］タブがあります。

モデル ナビゲータのタブは以下の表に示す順に配置されています。

優先順位	タブ名	カテゴリ
1	シンボル	プレゼンテーション
2	ダイアグラム	プレゼンテーション
3	リンク	リンク
4	パッケージ	エンティティ
5	特性	エンティティ
6	お気に入り	エンティティ
7	定義	エンティティ
8	ショートカット	エンティティ
9	参照	エンティティ
10	モデル インデックス	エンティティ
11	最近	エンティティ

ナビゲーション

選択肢をダブルクリックすると、[モデル ビュー] とダイアグラムの両方に選択肢が表示されます（可能な場合）。

CTRL キーを押しながらクリックまたはダブルクリックすると、再びモデル ナビゲータの焦点がクリックした選択肢になります。また、選択肢が [モデル ビュー] とダイアグラムの両方に表示されます（可能な場合）。

Shift キーを押しながらクリックまたはダブルクリックすると、選択肢は [モデル ビュー] のみに表示されます。ダイアグラムには表示されません。

モデル ナビゲータのタブとショートカット メニューに、最近使用したモデル ナビゲータのエンティティのリストが表示されます。このリストで、最近、現在のモデル ナビゲータ エンティティとして使用したエンティティを、再度モデル ナビゲータの焦点にできます。

プレゼンテーション タブ

シンボル

[シンボル] タブには、現在のエンティティに関連したシンボルとラインが表示されます。

ダイアグラム

[ダイアグラム] タブには、現在のエンティティと密接に関連したダイアグラムが表示されます。

リンク

[リンク] タブには、現在のエンティティの外部から、および、外部へのハイパーリンクのリストがあります。リンクをクリックして、そのリンクに関連付けられたリンクのエンド ポイントにナビゲートします。

エンティティ タブ

パッケージ

[パッケージ] タブには、現在のエンティティを含むパッケージで表示される定義がすべて一覧表示されます。

特性

現在のエンティティがクラスまたは類似したものである場合（正確には、現在のエンティティが分類子であるか、または分類子に含まれている場合）、[特性] タブにクラスの定義と継承された定義のリストが表示されます。

定義

［定義］タブでは、現在のエンティティのスコープのローカル定義と継承された定義がすべて一覧表示されます。

参照

［参照］タブには、定義が使用されている場所に素早くナビゲートできるよう、現在の定義タブへのモデル参照のリストが表示されます。このタブに含まれる情報は［モデル ビュー］のショートカット メニューの［参照の一覧表示］と類似しています。

ショートカット

［ショートカット］タブで、モデルの一般に利用される関係を素早くナビゲートします。最も一般的なショートカットについては、［ショートカット］カラムに関するテキストで説明します。

お気に入り

モデル内で再度ナビゲートしたい場所を選択するため、［お気に入り］タブでお気に入りの設定方法やナビゲートの方法を確認します。このタブの内容は、現在のツール セッションのみで維持されます。リストの項目を追加または削除するには、リストで［…の追加］（または ［…の削除］）、または、［すべてのアイテムの削除］行をクリックします。

［モデル ビュー］内のショートカット メニューから［お気に入り］を選んで、選択されている要素をこのリストに追加することもできます。

モデル インデックス

［モデル インデックス］タブには、名称未設定パラメータ（戻りパラメータ）以外のモデル定義がすべてアルファベット順にリストされます。［検索］ダイアログの説明も参照してください。

最近

［最近］タブで、モデル ナビゲータが焦点を合わせたエンティティをトラックして、最近使用したエンティティを再度モデル ナビゲータの焦点にできます。このタブの代りに、最近使用したモデル ナビゲータのエンティティを最大 5 つ表示するショートカット メニューを利用することもできます。

カラム

以下はモデル ナビゲータに表示されるカラムのリストと、表示される情報の簡単な説明です。

[インデックス] カラム

この列は、[参照] タブと [お気に入り] タブにあります。このカラムにはエンティティがアクセスされた順番を示す数字が表示されます。数字が小さいほど、最近アクセスされたことを意味します。

[リンク] カラム

現在のエンティティへの外部からのリンクと外部へのリンクの数。

[場所] カラム

モデル内の選択肢の場所

[名前] カラム、[ダイアグラム名] カラム

選択肢の名前

[ページ] カラム

ダイアグラムのページ番号。このカラムは、[ダイアグラム] タブにあります。

[ロール] カラム

参照リストにおける現在のエンティティのロールを示すリスト。このカラムは、[参照] タブにあります。

[ショートカット] カラム

このカラムには、現在のエンティティからさまざまな関連エンティティへのショートカットがリストされています。このカラムは、[ショートカット] タブにあります。以下に、[ショートカット] カラムに表示されるショートカットの例をいくつか挙げます。

- [スコープ] ショートカット：現在のエンティティを含むスコープ エンティティをモデル ナビゲータの焦点とします。
- [コンテナ] ショートカット：現在のエンティティを所有するエンティティをモデル ナビゲータの焦点とします。
- [モデル ルート] ショートカット：現在のエンティティのモデル ルートをモデル ナビゲータの焦点とします。特に、2 つ以上のモデルを持つワークスペースがある場合、このショートカットは便利です。
- [定義済みのパッケージ] ショートカット：定義済みタイプの内部ライブラリをモデル ナビゲータの焦点とします。

［種類］カラム、［アイテムの種類］カラム、［ダイアグラムの種類］カラム

記述されたエンティティのタイプ。たとえばクラス、シンボル、クラス図などがあります。

［ビュー］カラム

現在の定義を示すシンボルとラインの数。

ダイアグラムの生成

DOORS Analyst は、既存のモデル要素を可視化する目的でダイアグラムの自動生成をサポートします。一般的によく使用されるダイアグラム、たとえば継承図、合成図、依存関係図などを生成する、組込み済みのダイアグラムジェネレータが多数用意されています。特定のニーズのためにカスタムダイアグラムジェネレータを追加することもできます。

ダイアグラムを生成するには、以下の手順を実行します :

1. モデルビューで要素を選択します。選択した要素にしたがって、生成されるダイアグラムが決まってきます。たとえば、特定のクラスの上位クラス、下位クラスを可視化したい場合は、そのクラスを選択する必要があります。
1. コンテキストメニューから［ダイアグラムの生成］を選択して、サブメニューで使いたいダイアグラムジェネレータを選択します。たとえば、継承図を生成したい場合は、［Generate inheritance view］を選択します。

生成されたダイアグラムは通常は選択した要素の下に配置されます。ただし、一部のダイアグラムジェネレータはモデル内の別の場所に配置します。たとえば、最上位のダイアグラムとして配置したり、別パッケージ内に配置します。配置された後で、希望の場所に移動できます。

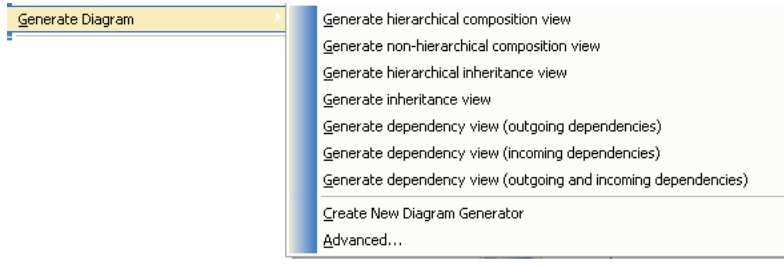

図 17: ダイアグラムの生成 コンテキストメニュー

ダイアグラム生成パラメータ

ダイアグラムジェネレータに実パラメータを与えてダイアグラムの生成を制御できます。たとえば、あるクラスの継承ビューを生成したい場合は、パラメータを使用して、直近の上位下位クラスのみを表示するのか、すべての上位下位クラスを表示するのかを制御できます。

ダイアグラムジェネレータをコンテキストメニューの［ダイアグラムの生成］から実行する場合、パラメータにはデフォルト値がセットされます。このパラメータを変更するには、生成されたダイアグラムで［ダイアグラム生成パラメータの編集］コンテキストメニューを使用します。生成されたダイアグラムからプロパティエディタを開

いて、<<generated>> ステレオタイプを選択し、パラメータを編集することもできます。［Parameters］フィールドでテキストとして編集することも、［Edit Parameters］ボタンを押してダイアログから編集することもできます。

ダイアグラムの再生成

生成されたダイアグラムは、モデルの新しい情報に基づいて再生成できます。たとえば、新しい上位下位クラスが追加されたときに継承図を再生成したい場合などに有用です。ダイアグラムコンテキストメニューの［再生成］を使用して、再生成できます。ダイアグラム生成パラメータを修正した場合にもダイアグラムを再生成する必要が生じます。

生成したダイアグラムを再生成するには、ダイアグラムのコンテキストメニューから利用可能な［再生成］コマンドを使用します。［ツール］メニューから［すべての大グラムの再生成］を選択して、モデル内のすべてのダイアグラムを再生成することもできます。このコマンドで再生成されるのは自動生成されたダイアグラムだけです。

重要！

ダイアグラムの再生成を実行すると、ダイアグラムに含まれていたすべてのものが削除されて再生成されます。もし自動生成したダイアグラムに手動で変更を加えていた場合、たとえば、レイアウトや色の変更を手動で行っていた場合は、それらの変更はすべて失われます。

生成されたダイアグラムを通常のダイアグラムに変更する

修正されたダイアグラムに対して誤って再生成を実行することを回避するために、手動で管理してゆくダイアグラムについては、自動生成ダイアグラムから通常のダイアグラムへと変換することを推奨します。このためには、以下の手順を実行します。

1. モデルビューで生成したダイアグラムを選択します。
2. コンテキストメニューから［ステレオタイプ ...］を選択します。
3. ‘generated’ チェックボックスのチェックを解除して［OK］を押します。
4. この操作を行うとこのダイアグラムは再生成できなくなります。

既存のダイアグラムでダイアグラムジェネレータを使う

ダイアグラムジェネレータは必ずしも新規のダイアグラムを生成するわけではありません。既存のダイアグラムに情報を追加するためにダイアグラムジェネレータを使うこともできます。このためには以下の手順を実行します。

1. 右マウスボタンを使用して、モデルビューから対象の要素をダイアグラムにドラッグします。
2. 要素をダイアグラムにドロップして、表示されるコンテキストメニューから［ダイアグラムの可視化］を選択します。
3. サブメニューで使用したいダイアグラムジェネレータを選択します。

要素をドロップした場所に、選択した ダイアグラムジェネレータで生成されたシンボルとラインが挿入されます。

高度なオプション

［ダイアグラムの生成］コンテキストメニューで表示されるダイアグラムジェネレータの他に、より高度なダイアグラムジェネレータも用意されています。これらのダイアグラムジェネレータを使うには、［ダイアグラムの生成］コンテキストメニューから［詳細 ...］コマンドを選択します。この操作で［ダイアグラム生成］ダイアログが開きます。

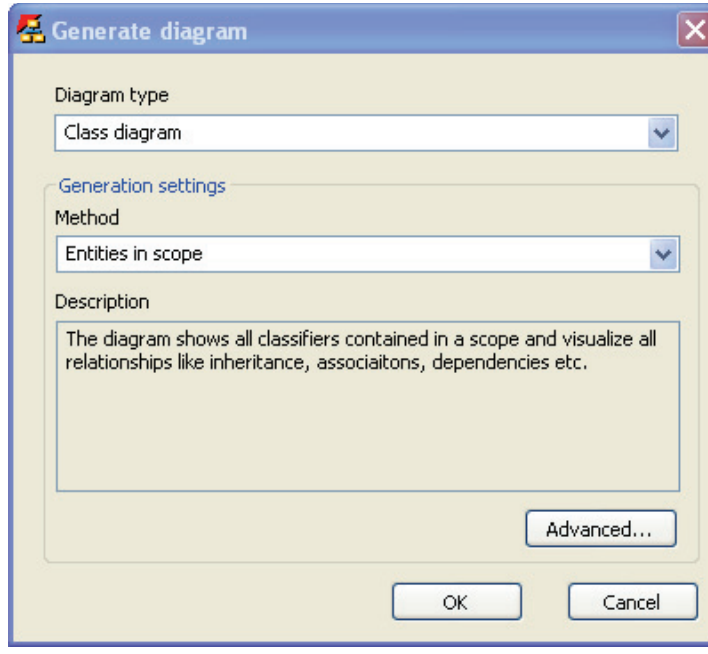

図 18: ダイアグラム生成ダイアログ

このダイアログでは、ダイアグラムジェネレータの種類は限定されますが、よりカスタマイズ可能なレイアウトオプションを指定できます。

ダイアグラムタイプ

［ダイアグラム生成］ダイアログでの最初の操作は、［ダイアグラム タイプ］を選択することです。ダイアグラムを生成するメソッドのあるダイアログタイプのみが表示されます。

生成の設定

次の操作は、生成方法である［メソッド］を選択することです。選択された［ダイアグラムタイプ］に適用できるメソッドのみが表示されます。

選択された生成メソッドの説明は、利用可能な生成方法のリストの下の［説明］欄に表示されます。

［詳細］ボタンを押すと、ダイアログが表示され、選択された生成メソッドについての詳しい設定ができます。これらの設定は生成されたダイアグラムに関連付けられ、生成後にプロパティエディタで編集できるようになります。

カスタマイズ

カスタムダイアグラムを生成する目的で、独自のダイアグラムジェネレータを作成できます。詳細については、ダイアグラムジェネレータの追加を参照してください。

プログラムからダイアグラムジェネレータを起動することもできます。これは、アドインを作成する場合などに有用です。詳細についてはダイアグラムジェネレータのプログラムからの起動を参照してください。

クエリ

このセクションでは、一定の条件を満たすエンティティを探すための UML モデルのクエリの実行方法について説明します。

クエリは、［検索］ダイアログの基本的な検索機能では探せないモデル内のエンティティを検索するときに便利な機能です。クエリは、必要な情報を探すための、標準 API の代替手法です。クエリでは、使用可能な多くの API 関数（COM、C++、Tcl）の呼び出しを行うので、ある 1 つのクエリの式の機能は該当する API セットを使用する場合と同等です。

概念

クエリは、モデルからのエンティティのコレクションを返す操作です。

「述語」は、ブール値の true または false を返す操作です。

クエリと述語はいずれも任意数の入力引数を取ることができます。また、常に暗黙的に存在する入力引数の 1 つとして、「モデル コンテキスト」があります。このモデル コンテキストというエンティティ上で、クエリまたは述語が呼び出されます。

UML モデル内で、クエリと述語操作を定義できるように、TTDQuery というライブラリが用意されています。これは 79 ページの図 19 のステレオタイプを定義します。82 ページの「［クエリ］ダイアログ」も参照してください。

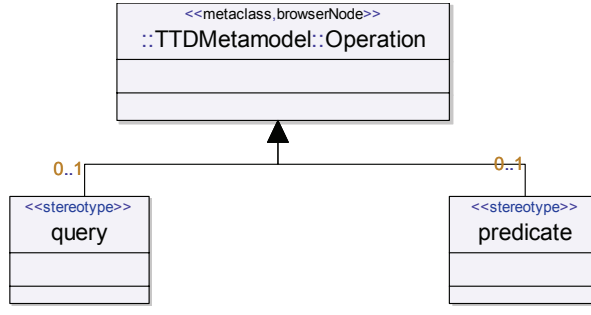

図 19: ステレオタイプを持つ TTDQuery ライブラリ

これらのステレオタイプに加えて、TTDQuery プロファイルには、すぐに使用できる多くのクエリと述語が用意されています。

クエリと述語の呼び出しは、クエリ式 にまとめられます。これは、DOORS Analyst によって解釈可能な式で、クエリ操作の実行時と同じように、解釈の結果としてモデルからのエンティティのコレクションが返されます。クエリ式では、ブール演算子とリテラルのほか、OCL にあるコレクション演算子の小サブセットも使用して、クエリと述語の呼び出しによって取得した結果を修正できます。

注記

公開 API の多くの操作は、クエリまたは述語として動作します。これらの操作は、クエリ式にも使用できます。これらの API 操作の UML 定義は、u2 と呼ばれるライブラリにあります。

クエリ式

クエリ式は、UML 式のテキスト構文で表します。クエリ式の型は、エンティティのコレクションです。つまり、クエリ式が解釈されると、結果はエンティティのコレクションとなります。

クエリ式に含まれるすべてのサブ式は、ブール型またはエンティティのコレクション型でなければなりません。ブール型の式の場合は、通常のブール演算子を使用できます。クエリ式では、以下のブール演算子とリテラルがサポートされます。

```
and (&&)
or (||)
not (!)
true
false
```

かっこ内の表記を使用することもできます。

エンティティのコレクション型の式の場合は、定義済みコレクション演算子を使用できます。

コレクション演算子

クエリ式内の式の実行から得られたエンティティのコレクションで、いくつかの定義済み演算子を使用できます。これらの演算子の名前と意味は、OCL（オブジェクト制約言語）に基づいています。実際、定義済みコレクション演算子を呼び出す場合に矢印表記（->）ではなくピリオド（.）を使用すること以外は、クエリ式は正式な OCL 式です。ただし Tau でサポートされるのは OCL のサブセットのみです。このサブセットを使用して、強力なクエリを実行できます。

select

構文：

```
select(<boolean expr>)
```

型：エンティティのコレクション

`select` は、エンティティの 1 つのコレクションをエンティティの別のコレクションへと変換します。結果のコレクションには、入力コレクションのエンティティのうちブール式が true と評価するものが含まれます。つまり、`select` は、述語を通してコレクションにフィルタをかけるために使用できます。

exists

構文：

```
exists(<boolean expr>)
```

型：boolean

Exists は、ブール式です。入力コレクション内に、少なくとも 1 つのブール評価が true になるエンティティが存在する場合は true を返し、それ以外の場合は、false を返します。

isEmpty

構文：

```
isEmpty()
```

型：boolean

この演算子は、入力コレクションが空の場合 true を返します。それ以外の場合は、false を返します。

例

使用可能な内蔵クエリと述語 を定義済みブール演算子とコレクション演算子と組み合わせて使用するクエリ式の例を以下に示します。

例 4

パッケージで定義されたすべてのアクティブ クラスを検索します。

[モデル コンテキスト = パッケージ]

```
GetAllEntities().select(IsKindOf("Class") and
HasPropertyWithValue("isActive", "true"))
```

例 5

クラスによって直接所有されているモデル内のすべての属性を検索します。

[モデル コンテキスト = モデル、つまりセッション]

```
GetAllEntities().select(IsKindOf("Attribute") &&
GetOwner().exists(IsKindOf("Class")))
```

例 6

モデル内のすべての <<access>> 依存を検索します。

[モデル コンテキスト = モデル、つまりセッション]

```
GetAllEntities().select(not
GetTaggedValue("access(..)").isEmpty())
```

このクエリによって必要な結果が取得されますが、かなり非効率的です。これは、モデル内のすべてのエンティティ上で、適用された <<access>> ステレオタイプのチェックが行われるためです。エンティティが「依存」であるというチェックを追加するだけでパフォーマンスが著しく向上します。依存以外のエンティティについて、GetTaggedValue クエリを呼び出す必要がなくなるからです。

```
GetAllEntities().select(IsKindOf("Dependency") and not
GetTaggedValue("access(..)").isEmpty())
```

HasAppliedStereotype 述語を使用して、式を書き直せます。これは、ステレオタイプが要素上で適用されているかどうかをチェックするときに推奨される方法です。

```
GetAllEntities().select(IsKindOf("Dependency") and
HasAppliedStereotype("access"))
```

<<access>> 依存を効率よく最短で検索するためには、クエリ式に、次のように GetStereotypedEntities クエリを使用します。

[モデル コンテキスト = TTDPredefinedStereotypes ライブラリ内の <<access>> ステレオタイプ]

```
GetStereotypedEntities()
```

この例のように、同じ結果を取得するために使用できるいくつかのクエリ式が存在する場合があります。同じ意味を持つクエリ間で実行パフォーマンスが大きく異なる可能性があるので、クエリ式を書く前に、それぞれの選択肢を検討することが重要です。

[クエリ] ダイアログ

[クエリ] ダイアログを使用して、実行するクエリ式を組み立てられます。このダイアログを開くには、[モデル ビュー] またはダイアグラムでエンティティを選択して、メニュー項目 [編集] -> [クエリ] を選択します。選択したエンティティがクエリ式のモデル コンテキストになります。

注記

クエリ式のモデル コンテキストとして、プレゼンテーション要素（ダイアグラム内のシンボルや行など）を使用できます。ダイアグラム内の選択されたエンティティから [クエリ] ダイアログを開くと、選択したプレゼンテーション要素がモデル コンテキストになります。モデル要素のクエリを実行する場合は、ショートカット メニュー [モデル ビューで表示] を使用して、対応する要素を [モデル ビュー] で検索します。

[クエリ] ダイアログには、現在のモデルで検索されたすべての使用可能なクエリと述語がリストされます。このリストには、定義 済み TTDQuery と u2 ライブラリで提供されるすべての「内蔵」クエリと述語が含まれます。また、他の場所で定義されているすべてのクエリと述語（ユーザー定義のクエリと述語など）も含まれます。

クエリ式を実行するには、[実行] ボタンを押します。デフォルトでは、結果が [検索結果] タブに出力されます。出力先は、ドロップ ダウン コントロールで別のタブ名を指定して変更できます。

編集コントロールで直接式を書くか、または使用可能なクエリと述語のリストでエントリをダブルクリックして、クエリ式を組み立てられます。選択した操作（クエリまたは述語）に入力仮パラメータがない場合、操作の呼び出しが直接クエリ式のテキス

ト内のカーソルの位置に追加されます。ただし、操作に 1 つ以上の入力仮パラメータがある場合は、ポップアップ ダイアログ（83 ページの図 20 を参照）が表示されます。このダイアログで、操作呼び出し用の対応する実パラメータを指定できます。

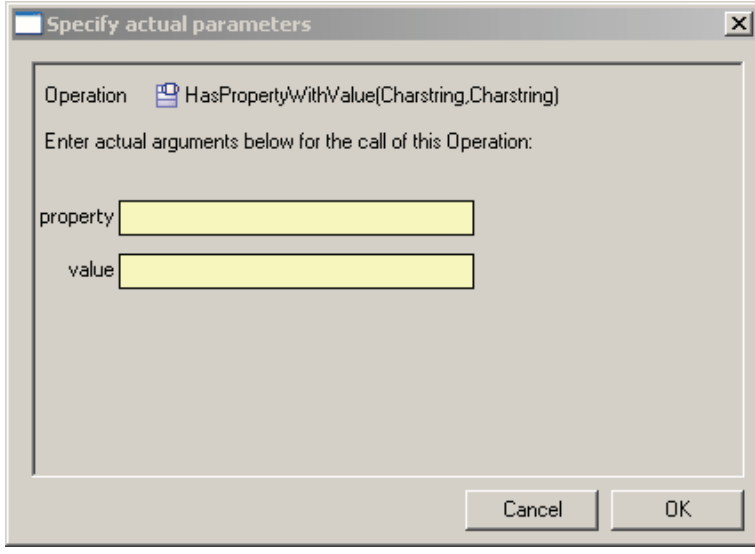

図 20 実パラメータの指定

このダイアログは、実際はプロパティ エディタ（パラメータは操作のプロパティと見なされる）であり、編集された値はプロパティ エディタと同じカラー コードに従います。パラメータの意味について「これは何？」ヘルプを表示するなど、プロパティ エディタの他の機能も利用できます。

クエリ式を新規クエリとして保存

［クエリ］ダイアログの［保存］ボタンを使用して、クエリ式をモデルの新規クエリとして保存できます。作成したクエリ式を今後も使用するために保存したい場合、この機能を使用します。新しいクエリの名前と説明、およびそのクエリを保存するモデル内の場所を指定するように指示されます。すべてのクエリを、別の .u2 ファイルに保存されるプロファイル パッケージなど、共通の場所に格納するとよいでしょう。これで、保存したクエリを複数のプロジェクトに組み込んで使用できます。

クエリ式を新規クエリとして保存すると、新しいクエリ式で使用可能なクエリと述語のリストに入り、ただちに利用できるようになります。

内蔵クエリと述語

クエリ式には、あらかじめツールに内蔵されたさまざまなクエリと述語を使用できます。これは、プロファイル ライブラリ TTDQuery と u2 で定義されて文書化されています。

また、ユーザー定義クエリと述語で説明したように、ユーザー定義クエリと述語を追加することもできます。

ユーザー定義クエリと述語

あらかじめツールに内蔵されているクエリと述語のほかに、新たなクエリと述語を定義できます。これを行うには、<<query>> または <<predicate>> ステレオタイプが適用されているエージェントを定義します。そのようなエージェントの実装では、クエリまたは述語のシグニチャの要件を満たす必要があります。したがって、クエリ エージェントは一連のエンティティを返し、述語 エージェントはブール値を返す必要があります。この必須出力パラメータは、最初のパラメータとしてエージェントに渡されます。また、エージェントは任意の数の入力パラメータを取ることができます。これらのパラメータには、エージェントによってサポートされている任意の型を使用できます。

API からのクエリ式の実行

84 ページの図 21 のエージェントを使用して、公開 API からクエリ式をプログラムにとして実行できます。

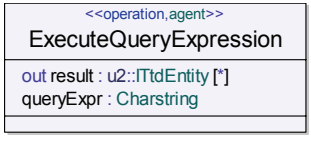

図 21 クエリ式のエージェント

このエージェントは（他のエージェントと同じように）、InvokeAgent 操作を使用して起動します。

例 7: Tcl API からのクエリ式の実行

以下の例は、Tcl スクリプトからクエリ式 “GetAllEntities()” を実行する方法を示しています。スクリプトによって結果のエンティティの Tcl ID が単に出力されます。

```
set s [std::GetSelection]
set a [u2::FindByGuid $U2 "@TTDQuery@ExecuteQueryExpression"]
set p [lappend p {} "GetAllEntities()"]
u2::InvokeAgent $U2 $a $s p
output [lindex $p 0]
```

ドラッグ アンド ドロップ

ここでは、ドラッグ アンド ドロップによるモデルの操作方法について説明します。

ドラッグ アンド ドロップ操作は、ドラッグ ソースとドロップ ターゲットとして以下の 3 つの組み合わせがあります。

- モデル ビュー内
- モデル ビューからダイアグラムへ
- ダイアグラム内とダイアグラム間

ドラッグ アンド ドロップ操作は、マウスの左ボタンまたは右ボタンを使用して実行できます。マウスの右ボタンを使用してドラッグ アンド ドロップ操作を行う場合はショートカット メニューが開き、ソース要素からターゲット要素へドラッグした結果として、実行可能な操作が表示されます。ショートカット メニューには必ず強調表示で示される選択肢があります。これは、マウスの左ボタンを使用してドラッグ アンド ドロップを行った場合に実行される操作を示しています。操作の隣にかっこ付きでモディファイア キーが表示されている場合もあります。この操作は、モディファイア キーを押した状態で、マウスの左ボタンを使用してドラッグ アンド ドロップを行うことで、実行が可能です。

次のセクションでは、ドラッグ アンド ドロップによって実行可能な操作について説明します。

モデル ビュー内

移動

モデル ビュー内の要素を移動します。

これは、モデル ビュー内でのドラッグ アンド ドロップのデフォルトの操作です。マウスの左ボタンを使用してドラッグ アンド ドロップを行った場合に、実行されます。

コピー

モデル ビュー内の要素をコピーします。

この操作は、Ctrl キーを押しながらマウスの左ボタンを使用してドラッグ アンド ドロップ操作を行った場合に、実行されます。

リンク

ドラッグ ソース要素とドロップ ターゲット要素との間にリンクを作成します。現在アクティブなリンク タイプが使用されます。

参照

リンクを使った作業

トレービリティを含むコピー

モデルビュー内の要素（サブ要素を含む）をコピーして、コピーからオリジナルへの <<trace>> 依存を作成します。

この動作を行わせるには、右マウスボタンを使ってドラッグアンドドロップし、［トレーサビリティを含むコピー］コマンドをポップアップメニューから選択します。

依存は、パッケージ、クラス、属性、操作などすべての定義について作成されます。

モデル ビューからダイアグラムへ

プレゼンテーションの作成

ドラッグ ターゲット要素との関連で、ドラッグ ソース要素を表すシンボルを作成します。

これは、モデル ビューからダイアグラムへのドラッグ アンド ドロップのデフォルトの操作です。マウスの左ボタンを使用してドラッグ アンド ドロップを行った場合に、実行されます。

プレゼンテーションの作成 (ラインを含む)

［プレゼンテーションの作成］と同じ操作ですが、ドロップ ターゲット ダイアグラムの他の要素とドラッグ ソース要素との接続を表すラインが作成されます。

ダイアグラムの可視化

ドラッグ ソース要素とドロップ ターゲット要素のために用意されているダイアグラム生成メソッドを含むサブメニューです。ドラッグソース要素は、ダイアグラム内の既存の要素に影響を与えずに、ダイアグラム内で可視化されます。

参照

ダイアグラムの生成

ダイアグラム内とダイアグラム間

ダイアグラム内とダイアグラム間でのドラッグ アンド ドロップによって実行される操作は、モデル ビュー内の場合と同じです。

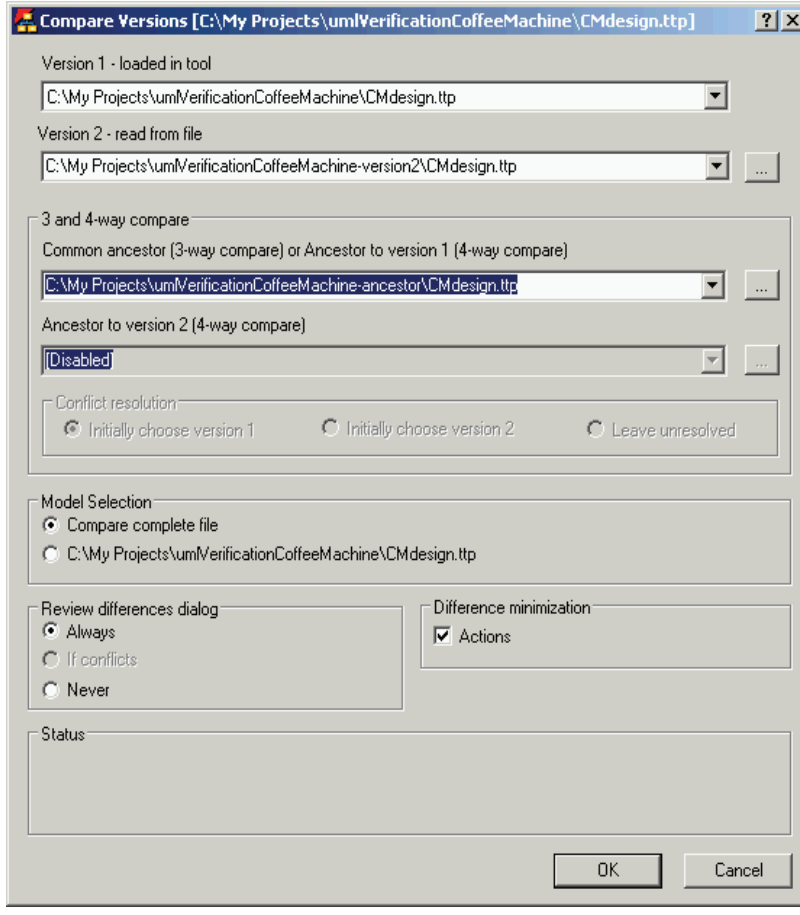

図 22: バージョン比較ダイアログ

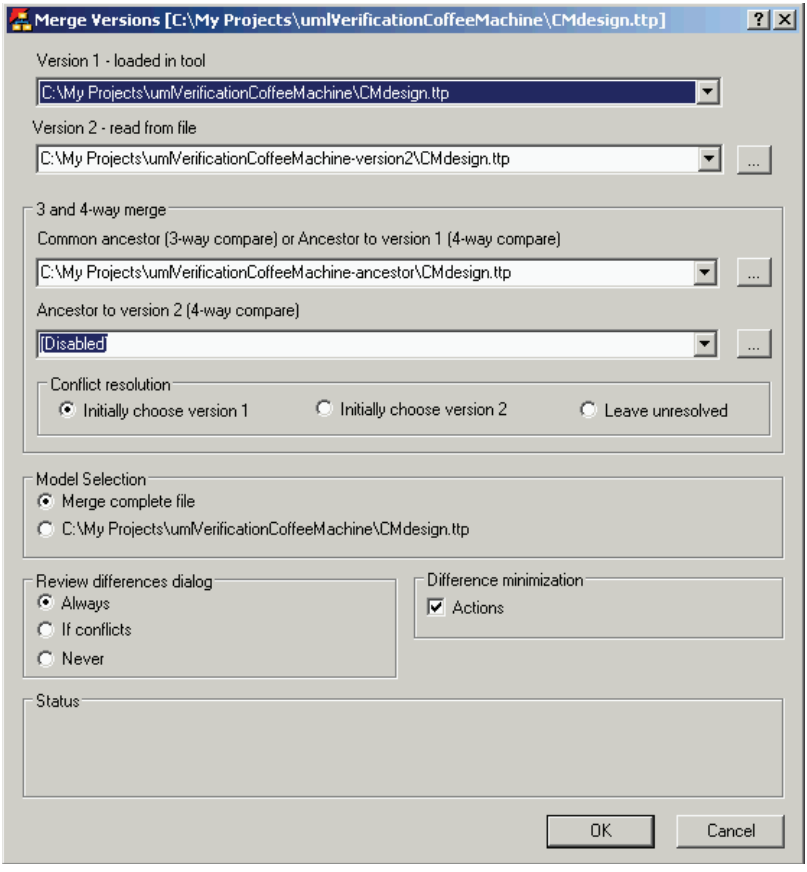

図 23: バージョンマージダイアログ

未解決のままにする：このオプションを選択すると、マージ操作は競合を未解決のままにします。どちらのバージョンを選ぶかを指定してすべての競合を明示的に解決するまで、マージ操作は完了しません。マージ操作は、一時的なバージョンを作成するために、バージョン 1 の前の世代（このバージョンは共通の世代（3 種類）またはバージョン 1 の前の世代（4 種類の比較）の指定値です）のプロパティを使用します。

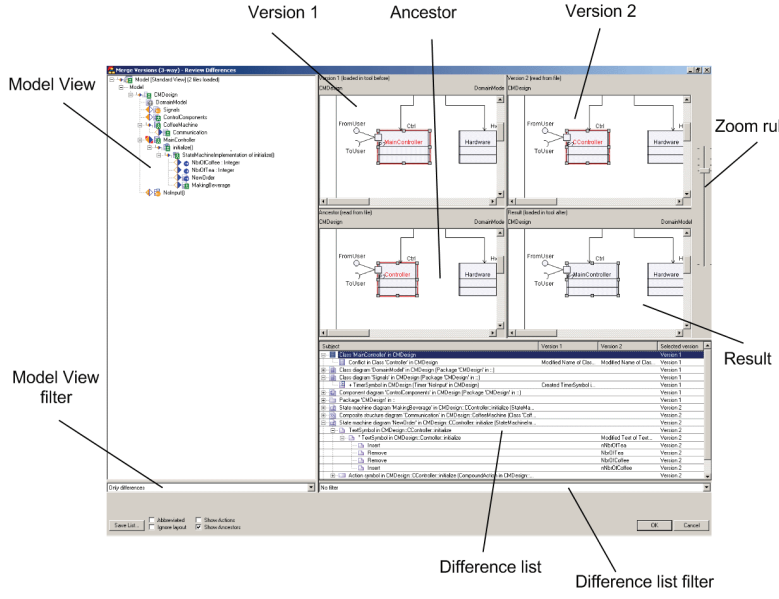

図 24: 相違点レビューダイアログ

External text compare/External text merge

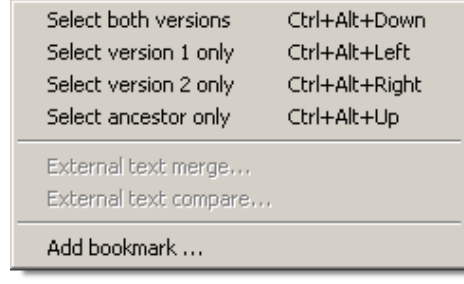

図 25: コンテキストメニュー

An external textual compare and merge tools can be used for comparing and/or merging comments, text symbols, task symbols and instance expressions. The “External text compare...” and “External text merge...” operations are available where applicable.

If an external textual merge is done, the result will be checked if it can be reentered into the model. If it cannot be entered into the model, the result file from the external tool is saved and the path is reported together with an error message box.

Path and command line switches for the external text compare/merge tool are available via the Tools menu, Options dialog, under the ［比較 / マージ］タブ tab.

Add bookmark

A bookmark can be added on a selected entity. A comment can be added to the bookmark. The bookmarks can later be listed in the model navigator.

生成されるイメージのサイズ、および未解析のテキストを保存するかどうかは、レビュー情報の保存オプションで指定できます。指定箇所は、［ツール］メニューから、［オプション］ダイアログを開き、［比較 / マージ］タブです。

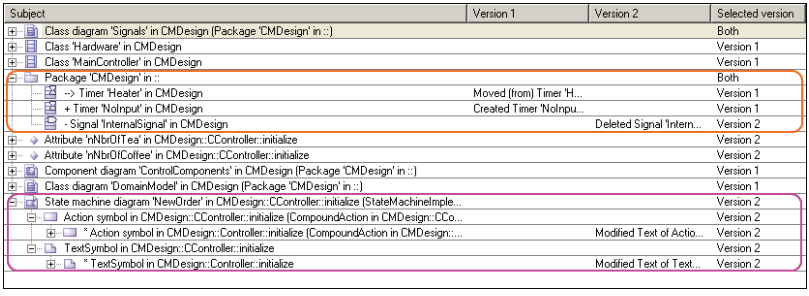

図 26: Semantic and presentation model differences grouping

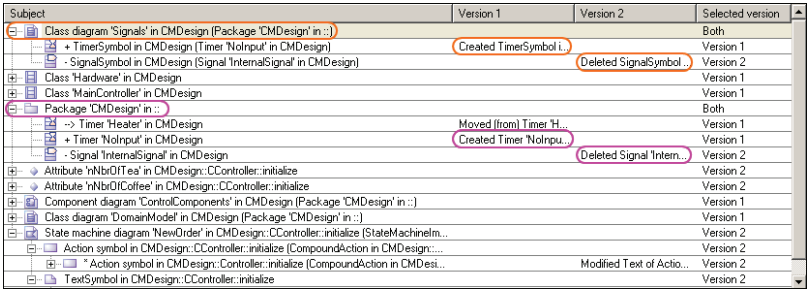

図 27: Grouping of “created entity” and “deleted entity” differences

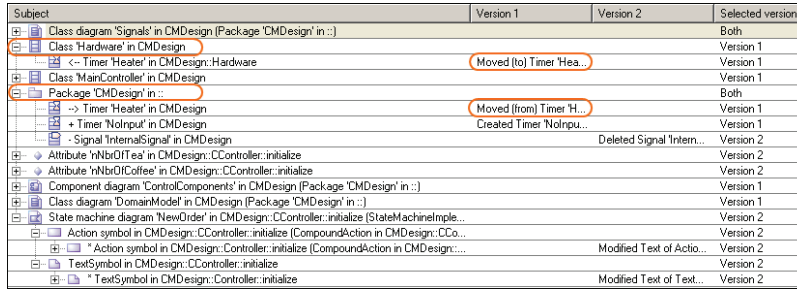

図 28: Grouping of “moved entity” differences

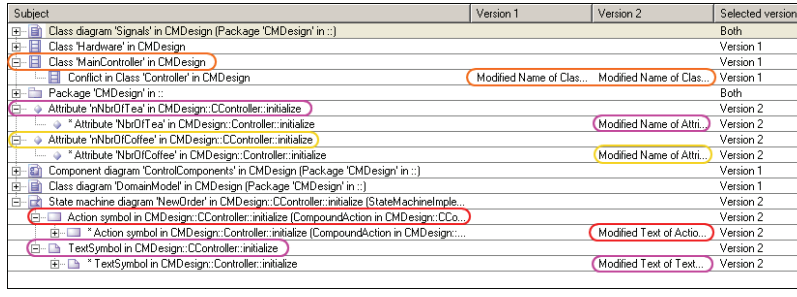

図 29: Grouping of “modified attributes” differences

This composite node contains conflicting differences which are owned by different representative elements. For example, if entity has been moved in Version 1 and in Version 2 and the new owners of that entity are different in Version 1 and Version 2, then the Composite Conflict Group will be created. This group can contain Difference Nodes only.

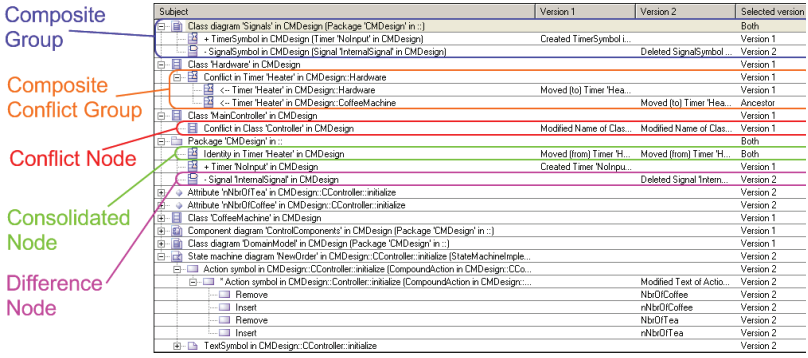

図 30: Nodes in Difference list

Conflict Node

This node corresponds to conflicting differences which are related to the same representative element.

Consolidated Node

This node corresponds to consolidated differences which are related to the same representative element.

Difference Node

This node describes the simple change that has been made in Version 1 or in Version 2.

Composite Textual Difference Node

This group node corresponds to a group of primitive textual differences. This group can contain Textual Difference Nodes only.

Textual Difference Node

This node corresponds to a primitive textual difference. There are two operations that represent modifications in the text: **Remove** and **Insert**. **Remove** means that a part of the text has been deleted (in comparison with the ancestor version). **Insert** means that a new text has been added.

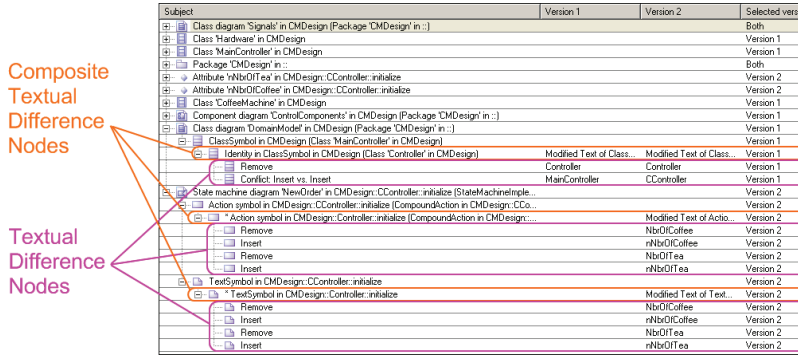

図 31: Textual difference nodes

When Textual Difference Node is selected in 相違点リスト , the modified part of code is highlighted in the one or several windows which represent Ancestor, バージョン 1, バージョン 2 and 結果 models, see 図 32 on page 94.

If the selected node corresponds to **Remove** operation, then the removed part of the text is selected in Ancestor window (and in 結果 window, if that operation is rejected). If the selected node corresponds to **Insert** operation, then the inserted part of the text is selected in バージョン 1 (and/or バージョン 2) window (and in the 結果 window, if that operation is accepted).

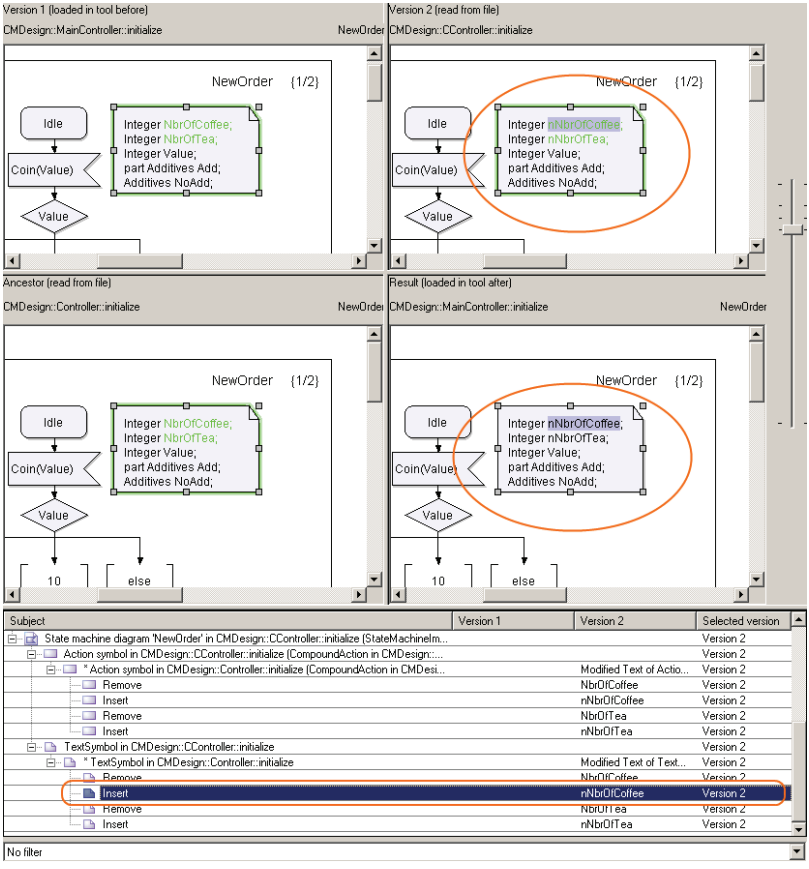

図 32: Textual difference highlighting

The colors are used in the 相違点ダイアログでのレビュー in order to simplify the understanding of changes that have been done in both versions. The **blue** color is used to mark presentation elements that correspond to model elements modified in version 1 only. The **green** color is used to mark presentation elements corresponding to model elements that have been modified in version 2 only. And presentation elements that correspond to model elements modified in both versions are marked by **red** color. The same coloring is applied to modified presentation model elements, i.e. Symbols and Lines as well as to parts of the text.

3

ダイアグラムの操作

プロジェクトを開いていれば、もうモデル編集の準備が整っています。

ダイアグラム エディタをモデル情報と組み合わせて使う場合、使い方は作成するアプリケーションによって異なります。以下は、ツールに慣れて、必要に応じてワークフローを自由に採用したり、変更できるようにするための推奨事項です。

ダイアグラムの一般的な操作方法

ダイアグラムの一般的な操作方法：

- ダイアグラムの作成
- ダイアグラムを開く、保存、印刷
- ダイアグラムの移動
- ダイアグラムのサイズ変更
- 検索
- テキスト解析
- ダイアグラムの自動レイアウト
- ビューの体系化

グリッド

ダイアグラム描画エリアにグリッドを表示します。グリッドが表示されると**グリッドに吸着**する機能が**オン**になります。グリッドの間隔は 2 ミリに設定され、変更できません。ショートカット メニュー、または、［オプション］の設定変更で、グリッドを表示または非表示にできます。デフォルトでは非表示です。シンボル、ライン、テキスト フィールド（シンボルに固定されたものを除く）は、すべてグリッドに吸着します。シンボルのサイズ変更もグリッドのメモリ単位に行われます。自動サイズ調整の場合も同様です。

フレーム

すべてのダイアグラムには、すべてのシンボルを内包するフレームがあります（ただしポート シンボルはフレーム上に配置）。キャンバスの左上隅から（x=10, y=10）ミリのところに、左上隅のフレーム シンボルが配置されます。フレームのサイズはダイアグラムのサイズとレイアウトに合わせて設定されます。フレームの上または外側にシンボルを置くため間隔を広げる必要があれば、サイズを変更できます。

フレームは、キャンバス スペースの許す限り、どの方向にもサイズ変更または移動できます。

ヘッダー

ダイアグラムのヘッダーは、ダイアグラムの左上隅に配置されます。テキストは右揃えです。テキストの位置は定義するエンティティのプロパティから算出されます。

ダイアグラム名

ダイアグラム名は、ダイアグラムの右上隅に配置されます。テキストは右揃えです。ダイアグラム名はモデル情報から算出されるため、［モデル ビュー］からのみ変更できます。

ダイアグラムの作成

ダイアグラムには、モデル要素を表示する一連のプレゼンテーション要素が含まれます。ダイアグラムはワークスペース ウィンドウから管理します。

1. **ワークスペース** ウィンドウで、[モデル ビュー] タブをクリックする。
2. 適切なパッケージを選択または作成する。
3. これで、ショートカット メニューから [新規] を選択して、適切なダイアグラム（または、モデル要素）を作成できるようになります。

参照

第 i 章「モデルの操作」の 68 ページ、「プレゼンテーションの作成」

ダイアグラムを開く、保存、印刷

プロジェクトを保存する際、ダイアグラムの情報がファイルに保存されます。特にファイルの指定がない場合、この情報はデータファイル（.u2 拡張子）に保存されます。このファイルは、通常、現在使用中のプロジェクト ファイル（.ttp）とワークスペース ファイル（.ttw）と同じ場所に保管されます。

- ダイアグラムを開くには：[モデル ビュー] のダイアグラム アイコンをダブルクリックする。
- 開いたダイアグラムを印刷するには：[ファイル] メニューから [印刷] をクリックする。

新しいダイアグラムの作成時、ダイアグラムのサイズはプリンタ設定から導き出されます。たとえば、プリンタの印刷の向きが横に設定されていれば、ダイアグラムも横に設定されます。

ダイアグラムイメージの保存

ダイアグラムは、JPEG、GIF、BMP、SVG などさまざまな形式の画像としてエクスポートできます。

ダイアグラムを画像として保存するには、ダイアグラムを開き、[ファイル] メニューの [名前を付けて保存] を選択して、表示されるダイアログで画像のファイル形式を指定します。

参照

第 8 章「印刷」の 346 ページ、「印刷するダイアグラムの選択」
第 11 章「ダイアログ ヘルプ」の 373 ページ、「保存」

ダイアグラムの移動

ダイアグラムは同じプロジェクト内、または、同じワークスペース内のプロジェクト間で移動できます。

- ［モデル ビュー］でダイアグラムに対応するアイコンをクリックし、このダイアグラムを目的の場所にドラッグする。

ダイアグラムのサイズ変更

ダイアグラムを作成すると、デフォルトで、自動サイズ調整モードに設定されます。ダイアグラムのショートカット メニュー、またはダイアグラム要素プロパティツールバーで、このモードのオン／オフを切り替えられます。ダイアグラムが自動サイズ調整モードに設定されていると、ダイアグラムの要素をエディタ キャンバスの好きな場所に、ドロップ、挿入、貼り付け、移動ができます。要素をフレームの外側に配置すると、ダイアグラムのサイズはその要素に合わせて自動的に変更されます。自動サイズ調整モードでは、常に、ページ全体が全ダイアグラム要素を収容できる最小値に設定されます。

ダイアグラムのサイズとレイアウトの初期設定は［印刷設定］（サイズと印刷の向き）で決定されます。プリンタがインストールされていない場合、サイズは現在の［オプション］設定で決定されます。ダイアグラムのサイズ変更は、以下の 2 つのステップで行います。ダイアグラムを拡大する場合、以下のいずれかの方法によって行います。

- ショートカット メニューでダイアグラムのサイズを変更する。［モデル ビュー］のダイアグラムを右クリックし、［ダイアグラム サイズ］を選択します。表示されたダイアログで手動によるサイズ変更を選択したり、自動サイズ調整モードに戻せます。
- 自動サイズ調整がオフの場合、Ctrl キーを押しながらフレーム シンボルのドラッグ ハンドルをクリックすると、ダイアグラムの用紙サイズ全体が段階的に拡大される。Ctrl+ Shift キーを押しながらダイアグラムのドラッグ ハンドルをクリックすると、サイズが縮小されます。押したキーに応じてマウス カーソルの形が変わります。

ダイアグラムのサイズに合わせて、フレーム シンボルのサイズが変わります。ダイアグラムのサイズを縮小する場合、フレーム シンボルが、ダイアグラム内のすべてのシンボル（またはライン）より小さくなることはありません。フレームは、内側のシンボル（ライン）から、1 グリッド ポイント以上離します。ダイアグラムのサイズを変更すると、ダイアグラム名シンボルは自動的に移動します。フレームに置かれたポートとラインはフレームの動きに連動して、移動します。キャンバスのサイズは、フレームと余白を合わせた大きさよりも小さくなることはありません。フレームのサイズ変更はグリッドの間隔で行われます。

検索

［編集］メニューから［検索］を選択して、［ダイアグラムと定義の検索］ダイアログを開きます。［ダイアグラムと定義の検索］ダイアログで定義を検索できます。定義が使用されている場所を検索するには、［ダイアグラム中のテキストも検索する］オプションを指定します。結果は出力ウィンドウの［検索結果］タブに表示されます。エンティティが使用されている場所を一覧表示するには、［モデル ビュー］のエンティティを右クリックし、ショートカット メニュー［参照の一覧表示］をクリックします。

参照

モデル インデックス

テキスト解析

一般に、テキスト シンボル（および外部ファイル）には C++ スタイル構文が使用されますが、他のすべてのシンボルには UML スタイル構文が使用されます。パーサは UML からのわずかな逸脱（「+」ではなく「public」、「^」ではなく「output」など）であれば受け入れますが、すべてを UML スタイルに逆構文解析します。テキスト シンボルでは、UML スタイルの逸脱は受け入れられますが、C++ スタイルに変換されます（可視性演算子の「+」は「public」に変換）。

これらの変換は以下のカテゴリに分類されます。

- 可視性：+ を `public` に変換
- 可視性：- を `protected` に変換
- 可視性：# を `private` に変換
- シグナル送信：^ を `output` に変換
- 分岐選択肢：`else` を `default` に変換

逆構文解析フェーズの他のプロパティもあります。

- パラメータ方向 “in”、逆構文解析されません。
- 引用符を必要としない名前であれば、引用符が付いた名前から引用符がなくなります。（逆構文解析後 `'Name1'` は `Name1` になります。
- パラメータ方向 `"in / out"` は、逆構文解析後、`"inout"` になります。
- リテラルを含むデータ型のみ列挙データ型になります。つまり、`datatype colors { literals red, green; }` は逆構文解析されて `enum colors { red, green }` となります。

逆構文解析でショートカットの表記を展開できます。一度に定義された複数の属性、リモート変数、シグナル、タイマー、例外または同義語（たとえば、`Integer i, j, k;`）は、逆構文解析後、複数の個別の定義（`Integer i; Integer j; Integer k;`）に展開されます。

アンパーサは、シグナルやタイマーの定義で省略されていた丸かっこを追加します（`"timer T"` は `"timer T()"` になります）。

範囲開始値は（可能な場合）、次の UML スタイルに変換されます：`">= n"` は `"n..*"` に変換され、`">=0"` は `"0..*"` に変換されます。

自動引用符付け

自動引用符付けの目的は、引用符 の入力を支援することです。名前に空白を入れる場合は、その両側に引用符を追加する必要があります。自動引用符付けが適用されるのはごく一部のシンボル（ラベル）のみです。たとえば、名前が含まれているラベルなどは、自動引用符付けされます。

ワード ラップ

ワード ラップでは、語を複数の行に分割できます。この機能は、複数行ラベルと自動サイズ調整されないシンボルに適用されます。語の分割箇所を決定するため、次の文字や符号が検索されます。「:」、「::」、空白、大文字、カンマ（「,」）、ピリオド（「.」）、アンダスコア。

ダイアグラムの自動レイアウト

ダイアグラム（キャンバス背景）のショートカット メニューには、自動レイアウト アルゴリズムが関連付けられている、ダイアグラム タイプ用の、［自動レイアウト］メニュー項目があります。メニュー項目を選択すると、ダイアグラム要素は、特定のダイアグラム タイプに適したレイアウトに配置されます。たとえば、クラス図と状態機械図には階層的なレイアウトがあります。

［要素の表示］ダイアグラムを使用してダイアグラム要素を配置する際、自動レイアウト アルゴリズムが使用されます。

自動レイアウトを使用する際、以下のことを考慮する必要があります。

- クラス図のレイアウト アルゴリズムに含まれるのは汎化ラインだけ。
- 状態機械図のレイアウト アルゴリズムにはフロー ラインと遷移ラインが含まれる。

ビューの体系化

エディタには、ビューを体系化するいくつかの機能があります。これらの機能にはスクロール、ズームイン／アウトなどのショートカットが含まれます。

ダイアグラムを閉じるときに、現在のスクロールとズームの設定を個別のファイルに保存できます。ファイルの拡張子は .u2x、名前は <project>_DiagramSettings です。このファイルはプロジェクト（ttp ファイル）に追加されません。プロジェクトのロード時に、拡張子 .u2s と対応する名前を持つファイルをロードするステップがあります。この機能は、［スクロールとズームの設定を記憶する］オプションで設定できます。

スクロール

ダイアグラムが、デスクトップに全体表示できないサイズに設定されている場合、ビューをスクロールできます。スクロールにはウィンドウ スクロール バーを使用します。

（**Windows** の場合）インテリマウス ポインティング デバイスでスクロールすることもできます。

- スクロール ホイールを使用して、垂直方向にスクロールする。
- Ctrl キーを押しながらスクロール ホイールを使用して、水平方向にスクロールする。

ズーム

［表示］メニューの［ズーム］コマンドを使用して、固定倍率で段階的に拡大／縮小できます。

ショートカット メニューで連続ズームを実行できます。ダイアグラムを右クリックし、［ズーム］にカーソルを合わせて、目的の拡大レベルを選択します。テキスト編集モードではない場合、マイナス記号（-）記号を使用してズームアウト、プラス記号（+）記号を使用してズームインできます。

（**Windows** の場合）インテリマウス ポインティング デバイスでスクロールできます。

- Shift キーを押しながらマウスの中央 ボタンを使用して、ダイアグラムをズームする。
- スクロール ホイールをダブルクリックして、等倍表示にする。
- Shift キーを押しながらスクロール ホイールをダブルクリックする。現在開かれているダイアグラム（カレント ダイアグラム）全体がデスクトップに表示されるようサイズが調整されます。

参照

ウィンドウのドッキング

ワークスペースの操作

DOORS Analyst のコマンド

このセクションでは、DOORS Analyst ウィンドウと DOORS の同期を取るために使用するいくつかのコマンドについて説明します。これらのコマンドは、以下のように 3 とおりの方法で表示されます。

- DOORS Analyst レイアウトの特別なツールバーのボタンとして表示される。
- Analyst というメニュー内に表示される。
- ダイアグラム内のオブジェクトを右クリックしてショートカット メニューからコマンドを選択して表示する。

レイアウトの切り替え

デフォルトの基本レイアウトでは、ダイアグラムと最も一般的な編集用のツールバーが表示されます。［レイアウトの切り替え］ボタンを使用して、ワークスペース ウィンドウ（および［モデルビュー］）と他のツールバーを含む表示と切り替えることができます。

DOORS Analyst を最初に起動すると、編集に使用できる一連のシンボルが［Analyst ビュー］という名前の構成（メタモデル）によって設定されます。これにより、通常使用するシンボルが見えるようになります。

要素の表示

DOORS Analyst でダイアグラムを作成する場合、［要素の表示］ボタンを使用して（ダイアグラムで右クリックして表示されるショートカット メニューからも使用できます）ダイアログを表示し、現在のダイアグラムに挿入する要素を選択できます。要素は自動レイアウト機能によって配置されます。

DOORS での変更の受け入れ

このボタンを使用して、DOORS Analyst の内容をフォーマル モジュール内の DOORS オブジェクトに反映させることができます。この機能は、DOORS フォーマル モジュールと DOORS Analyst の UML ビューの両方で平行して作業を行っている場合、DOORS での変更を UML ビューに反映させるために便利です。

DOORS での変更の確認

このコマンドを使用して、フォーマル モジュール内の DOORS オブジェクトの変更をチェックできます。この機能は、DOORS フォーマル モジュールと DOORS Analyst の UML ビューの両方で平行して作業を行っている場合に便利です。

DOORS で編集

このボタンを使用して、UML 要素から DOORS モジュール内の対応する要素に作業対象を切り替えることができます。

DOORS との同期を有効にする

ダイアグラムの 1 つまたは複数のシンボルを選択した状態でこのコマンドを使用すると、選択したシンボルと DOORS の同期を有効にできます。つまり、次に同期が取られたとき、DOORS フォーマル モジュールに対応するオブジェクトが作成されます。DOORS フォーマル モジュールの「Object Type」カラムには「Other」という文字が表示されます。これは、このオブジェクトが DOORS Analyst の通常の定義済みモデル要素の 1 つではないことを示します。

これらのオブジェクトの同期は一方向です。つまり、これらのオブジェクトに対する変更はダイアグラム自体で行う必要があります。DOORS フォーマル モジュールの変更をこれらのオブジェクトのダイアグラムに反映させることはできません。

DOORS との同期を無効にする

ダイアグラムの 1 つまたは複数のシンボルを選択した状態で、このコマンドを使用すると、選択したシンボルと DOORS の同期を無効にできます。ただし DOORS Analyst の定義済みモデル要素は DOORS と同期されます。

参照

5 ページの「DOORS における DOORS Analyst コマンド」

共通のシンボルの操作

シンボル情報

［シンボルとラインのツールチップを表示］を選択すると、ステレオタイプやバインド情報などのコンテキスト モデル情報を表示できます。

［表示モードのツールチップを表示］を選択すると、テキスト編集モードで構文解析情報を表示できます。

モデル要素の詳細の表示 / 隠すツールバー

［モデル要素の詳細の表示 / 隠す］ツールバーを使用して、ダイアグラム内のシンボルの特定機能の表示 / 非表示を切り替えられます。これらの設定はダイアグラムごとに保存され、ダイアグラム内のすべての要素に同じ設定が適用されます。

- Show/Hide qualifiers

  ラベル テキストの修飾子部分を切り替えます。たとえば、次のようになります。Package1::Package2::Class1 will be toggled to Class1.
- Show/Hide stereotypes

  ステレオタイプ ラベルの表示 / 非表示を切り替えます。ラベルを非表示にすると、ラベルが占有するスペースが最小と見なされ、結果的にシンボル サイズに影響する場合があります。
- Show/Hide quotation marks

  自動引用符の表示 / 非表示を切り替えます。これによって、一部のシンボルが自動引用符付けされ、このボタンの影響を受けます。引用符を非表示にしても、テキストはそのまま引用符付きと見なされます。自動引用符付けされるテキストは、通常の場合空白が含まれている名前です。

シンボルの追加

シンボルを追加するには、［ダイアグラム要素の作成］ツールバーの対応するアイコンをクリックしてから、ダイアグラム内でクリックまたは右クリックしてシンボルを配置します。

［モデル ビュー］からシンボルを生成することもできます。この場合、モデル要素を目的のダイアグラムにドラッグします。

状態機械図は、Ctrl キーを押しながら、新規シンボルの［ダイアグラム要素の作成］ツールバーをクリックしてフローへのシンボルの挿入を行えます。このシンボルは、現在選択されているシンボルの後に挿入されます。

既存要素を参照

ほとんどのシンボルについて、シンボルを右クリックするとショートカット メニューが表示されます。このメニューを持たないシンボルを以下に示します。

- 遷移ライン（状態（ステート）指向ビューでデザインされた場合、状態機械図で使用）
- ステート、シグナルおよび操作に関連付けられていない状態機械遷移シンボル

ショートカット メニューには以下のように、［新規 < モデル要素 > の作成］、［未接続を維持］、［既存要素を参照］の 3 つの選択肢があります。

- **新規 < モデル要素 > の作成**：新しいシンボルが作成され、シンボルに対応するモデル要素がモデルに作成されます。
- **未接続を維持**：新しいシンボルが作成されますが、これに対応するモデル要素は作成されません。
- **既存要素を参照**：タイプとスコープに合致する既存のモデル要素がドロップダウン ボックスに表示されます。

自動配置

前回のシンボルと接続して、シンボルを配置する場合が多くあります（たとえばクラスのポートなど）。このような配置を行うために、シンボルの自動配置機能があります。

- Shift キーを押したまま、シンボル ツールバーをクリックする。クリックしたシンボルは、現在選択しているシンボルに接続されます。
- Ctrl キーを押したままシンボル ツールバーをクリックする。クリックしたシンボルは、現在選択しているシンボルと次のシンボルの間に挿入されます。

シンボルが現在選択されているシンボルに対して、構文上正しいフローで接続されない場合、これらのシンボルはツールバーでグレー表示されます。

Shift + スペースキーおよび Ctrl + スペースキーを使用して、自動配置可能なシンボルのリストを表示することもできます。

参照

第 i 章「モデルの操作」の 39 ページ、「名前のサポート」
第 i 章「モデルの操作」の 68 ページ、「プレゼンテーションの作成」
第 i 章「モデルの操作」の 69 ページ、「モデルのナビゲートと作成」
ダイアグラムの自動レイアウト
要素の表示
メッセージの作成

要素の表示

［要素の表示］ダイアログで、現在のダイアグラムに表示するモデル要素を選択します。

［要素の表示］は以下のメニューから選択できます。

- ［ツール］メニュー
- ダイアグラムのショートカット メニュー

［要素の表示］を選択すると、現在のダイアグラムのシンボルとして表示可能なモデル要素のリストが表示されます。モデル要素にチェックマークを付けるか解除して、これに対応するシンボルをダイアグラムに追加あるいは削除できます。

［要素の表示］ダイアログには以下の機能があります。

- ［すべて］ボタンを 1 回クリックして、リスト内のすべてのモデル要素にチェックマークを付ける。
- ［なし］ボタンを 1 回クリックして、チェックマークをすべて解除する。
- ［短く表示］チェック ボックスで、モデル要素の概要リストと詳細リストを切り替える。
  - **概要リスト**には、現在のダイアグラムで一般的に使われるモデル要素が含まれます。（たとえば、クラス図のクラスなど）一般的なシンボルでなくても、すでに現在のダイアグラムにシンボルとして表示されているモデル要素は概要リストに含まれます。
  - **詳細リスト**には、選択されたスコープにある、現在のダイアグラムのシンボルとして表示可能なモデル要素がすべて表示されます。このリストには、現在のダイアグラムで一般的に使われるモデル要素のほか、特殊な変換も含まれます。（たとえば、ユース ケース図にアクターとして表示されるクラスの選択肢も詳細リストに含まれます。）
- ［スコープの選択］ボタンをクリックして、ダイアログを表示する。このダイアログからスコープを選択してメイン ダイアログに表示するモデル要素を選択します。デフォルトで、ダイアグラムが属するローカル スコープのモデル要素だけがこのリストに含まれます。

モデル要素は、ダイアグラム内の最後のプレゼンテーション要素（たとえばシンボルやラインなど）が削除されると自動的に削除されます。

シンボルの選択

Ctrl キーを押しながらテキスト フィールドの外側（シンボル枠の内側）をダブルクリックすると、シンボル、外向きラインと接続されたすべてのシンボルを選択します。

シンボルまたはライン以外の場所でクリックしてドラッグすると、選択矩形が作成されます。この矩形の中にあるものがすべて選択されます。

Ctrl キーを押しながら、シンボルまたはライン以外の場所でクリックしてドラッグしても、選択矩形が作成されます。この場合、矩形に接触したものがすべて選択されます。

状態機械 フローでは、Ctrl キーを押しながら、フローのシンボルをダブルクリックすると、該当シンボルと後続のシンボルすべてが選択されます。フローが分岐している場合も、この機能を利用できます。

シンボルの移動

シンボルを移動するには、シンボルをクリックしてダイアグラム内の希望する場所にドラッグします。シンボルを他のダイアグラムにドラッグすることもできます。

テキスト フィールドのあるシンボルの選択では、テキスト編集モードにならないようにします。テキスト編集モードではカーソルの形状が変わります。

テキスト フィールドの移動

いくつかのテキスト フィールド（ラベル）を移動できます。具体的には、ラインに属すラベルと、そのラベルがシンボル境界の外にあるシンボル（ポート、ピンなど）に属すラベルを移動できます。

この操作を実行するには、最初にラベルを選択します。その後にラベルをいずれかのハンドルでドラッグできます。新しい位置は、デフォルト位置のオフセットとして保存されます。ラベルが属すラインまたはシンボルを移動すると、ラベルも移動してオフセットが保持されます。

オフセットを解除するには、ショートカット コマンド［すべてのラベル位置のリセット］をクリックします。現在選択されているシンボルに属するすべてのラベルがそのデフォルト位置にリセットされます。

ラベルはデスクトップ上の任意の位置にドラッグできます。フレーム シンボルやダイアグラム領域の外の場合でも有効です。ダイアグラムの外のラベルは出力されません。

シンボルのサイズ変更

シンボルのサイズを手動で変更するには

1. 該当するシンボルを選択します。
2. 8 つあるグレーの正方形のうち 1 つにマウスのカーソルを合わせます。
3. マウスで、シンボルを目的の大きさまでドラッグします。

自動サイズ変更

すべてのシンボルで、［自動サイズ変更］を選択して中に入力したテキストのサイズにあわせてシンボルのサイズを自動変更できます。シンボルを右クリックして、ショートカット メニューから［自動サイズ変更］を選択します。

シンボルを折りたたむ

コンパートメント（クラス シンボルなど）のあるシンボルは、そのシンボルのショートカット メニューから［折りたたむ］メニュー項目をチェックして折りたたむことができます。コンパートメントとその中のラベル類は折りたたんだ状態では表示されません。

サイズ変更後のシンボル表示

シンボルの右下隅の外側に 3 つの点が表示された場合、シンボルのサイズが小さすぎてシンボルのテキスト フィールドにあるテキストをすべて表示できないことを意味しています。テキストに合わせてシンボルをサイズ変更するには、シンボルを選択して 3 つの点をダブルクリックします。

シンボルの接続

シンボルを手動で接続するには

1. シンボルをクリックして、ライン ハンドルを見つけます。
2. ラインを他のシンボルにドラッグします。
3. 移動先のシンボルに到達すると、ラインの先端に十文字の付いた丸が表示されます。シンボルの内側のラインを接続する境界に近い位置でクリックして、接続を完了します。

接続の結果、モデル要素となる場合もあります。たとえば、汎化ハンドルをクラス図の他のクラスにドラッグすると、2 つのクラス間のラインが作成され、同時に［モデル ビュー］に新しいアイコンが作成され、汎化が追加されたことが示されます。

状態機械図のシンボルは、自動配置でフローに自動的に接続されます。

リンクツールバーから 依存リンク の追加を行えます。

参照

ラインの描画

シンボル フローの編集

フローまたはフロー ブランチの選択

アクティビティ図のフローで、Ctrl キーを押しながらフローのシンボルをダブルクリックすると、該当シンボルと後続のシンボルがすべて選択されます。フローが分岐している場合も、この機能を利用できます。

フローへのシンボルの追加

アクティビティ図にシンボルを追加する際、接続されたシンボルのフローを作成できます。Shift キーを押しながらツールバーをクリックします。クリックしたシンボルは、現在選択しているシンボルに接続されます。シンボルが現在選択されているシンボルに対して、構文上正しいフローで接続されない場合、これらのシンボルはツールバーでグレー表示されます。

参照

自動配置

フローへのシンボルの挿入

フロー ラインまたは遷移ラインが選択された状態で C t r l キーを押すと、フローに挿入できるシンボルのみ、シンボル／ライン作成ツールバーで選択できる状態になります。

以下のいずれかを選択した状態で、Ctrl キーを押しながらボタンをクリックすると操作を挿入できます。

- シンボルを 1 つ選択した場合（このシンボルの後に操作が挿入されます）。
- ラインを 1 本選択した場合（このライン上に操作が挿入されます）。
- 1 つのラインで結ばれた 2 つのシンボルを選択した場合 (シンボル間に操作が挿入されます)

注記

- Ctrl キーを押しながら分岐シンボルを選択することはできません。分岐シンボルからは複数の出力フローが可能であるためです。

参照

自動配置

フローからのシンボルの削除

シンボルがフローから切り取り、または、削除された場合、削除されたシンボルとその接続ラインが自動作成ラインに置き換わります。

シンボルのテキスト フィールドの編集

シンボルのテキストフィールドを編集するには、まずシンボルを選択する必要があります。

- テキスト フィールドを編集するには、シンボルを選択して、そのテキスト フィールドの追加または変更したい場所をクリックします。これで、テキストを変更できます。ステレオタイプ情報など <<>>（ギルメット）で囲まれたテキストは編集できません。
- シンボルを選択してテキストフィールド内でダブルクリックすると、最も近くにあるテキストが選択されます。
- シンボルを選択してテキスト フィールドをクリック アンド ドラッグすると、編集モードに入って、テキストを選択できます。シンボルが選択されていないと、この操作でシンボルは移動します。
- シンボルを選択して F2 キーを押すと、シンボル内のメイン テキストを編集できます。テキストが 1 行の場合すべてのテキストが選択されますが、テキストが複数行に渡る場合は、テキストは選択されず、テキストの末尾にテキスト カーソルが表示されます。

注記

テキスト フィールドの外側（シンボル外枠の内側）をダブルクリックすると、シンボルのダブルクリック操作が実行されます（通常はナビゲーション）。

ダイアグラム要素のプロパティ

［ダイアグラム要素のプロパティ］と呼ばれるツールバーがあります。このツールバーには、選択したシンボル／ラインのさまざまなプロパティを制御するためのドロップダウン メニュー ボックスがあります。

- フォント
- フォント サイズ
- シンボル／ラインの背景色

このツールバーには、プロパティの設定を削除して、デフォルト設定に戻すボタンがあります。

シンボルを選択していない場合、ツールバー コマンドは現在のダイアグラムのすべてのシンボルに適用されます（個別にプロパティが設定されているシンボル以外）。

コメントの処理

コメント シンボルはすべてのシンボルに追加できます。

1. ツールバーのコメント シンボルをクリックします。
2. ダイアグラムにシンボルを配置します。
3. コメント シンボルの注釈ラインを、コメント添付先のシンボルに接続します。

コメントと制約

シグニチャシンボルのショートカット コマンド［コメントを表示］（［表示／非表示］のサブメニュー）を使用して、現在のダイアグラム内にコメント シンボルがないシグニチャ シンボルが所有するコメント モデル要素ごとに、1 つのコメント シンボルを作成して追加します。

シグニチャシンボルのショートカット コマンド［制約をシンボルで表示］を使用して、現在のダイアグラム内にシンボルがないシグニチャ シンボルが所有する制約モデル要素ごとに、1 つの制約シンボルを作成して追加します。

備考カラム

2 つ以上の コメント シンボルが近接して垂直またはほぼ垂直に配置されている場合、備考カラム が形成されます。112 ページの図 1 を参照してください。備考カラムが検出されると、垂直位置が自動調整されて、左揃えカラムになります。

Shift キーを押しながら最上部のコメント シンボルを垂直に少し（カラム幅全体を超えない範囲）移動すると、カラムを水平に移動できます。カラム内の別のコメント シンボルを少し移動すると（Shift キーを押しながら）、カラム内の所定の位置に戻されます。コメント シンボルを大きく移動すると、カラムから削除されます。

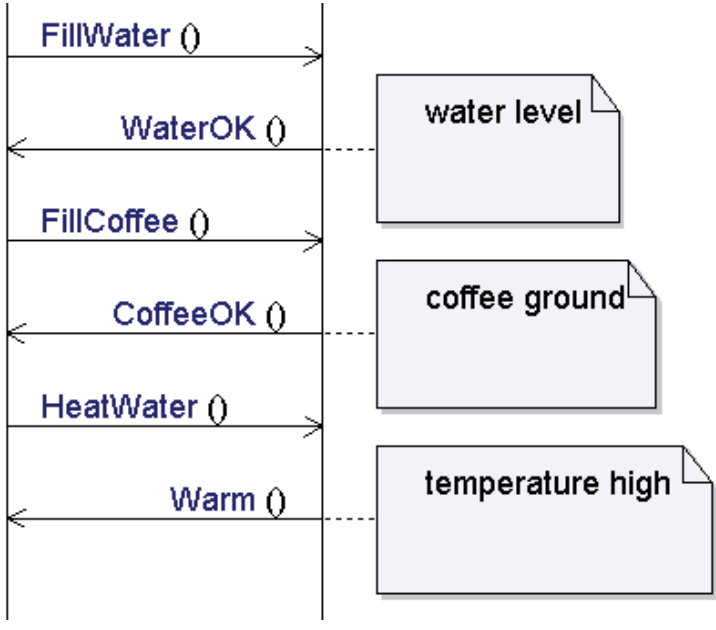

図 1: 備考カラム

注記

備考カラム内の最上部のコメント シンボルは、カラムにインクルードするライフライン ヘッダーの下に配置する必要があります。

シンボルのコピー、切り取り、削除、貼り付け

すべてのシンボルにはショートカット メニューがあります。このメニューは、シンボルを右クリックして表示できます。必要に応じて、このメニューから［切り取り］、［コピー］または［貼り付け］を選択します。

MS Word などの他のツールにシンボルを貼り付けることもできます。

シンボルを削除するには、削除するシンボルを選択して、Delete キーを押します。

注記

シンボルのタイプとモデルとの関係によって、シンボルの削除操作がモデルに影響を与える場合と、与えない場合があります。通常、ダイアグラムにシンボルを追加すると、モデルに情報を追加できます。シンボルとモデルに 1 対 1 の関係がある場合、ダイアグラムのシンボルを削除すると、モデル内の情報だけ削除できます。状態機械のフロー シンボルなどがこれに当てはまります。シンボルとこれに対応するモデル要素を削除するには、［モデルからの削除］を実行します。

アイコン

ユーザー指定アイコン

選択したシンボル アイコンを、画像ファイルを使用してユーザー指定アイコンに置き換えられます。アイコンは以下のレベルで指定できます。

- 特定シンボル
- 特定セマンティック モデル要素。モデル要素に関連づけられるすべてのシンボルが、このアイコンを使用します。
- 特定ステレオタイプ。このシンボルでステレオタイプ化されたモデル要素に関連づけられるすべてのシンボルが、このアイコンを使用します。
- 特定のタイプ（クラスやデータ型など）。このタイプのインスタンスに関連付けられるすべてのシンボルが、このアイコンを使用します。

ステレオタイプの追加

この機能はステレオタイプによって制御されます。この機能を使用するには、アイコンを持つモデル要素を右クリックし、ショートカット メニューから、［ステレオタイプ］を選択します。ダイアログで、ステレオタイプ［TTDStereotypeDetails::Icon］を選択します。プロパティ エディタから［ステレオタイプ］ボタンを使用して、ダイアログを開くこともできます。

このステレオタイプが適用できるエンティティは、メタモデル プロパティによって制御されます。この情報を参照するには、[モデルビュー] の [ライブラリ] セクションから、パッケージ [TTDStereotypeDetails] を開きます。そのクラス図で、サポートされるエンティティ（メタクラス）とアイコン ステレオタイプの関連を参照できます。

順序付け

上記の選択肢で、複数のユーザー指定アイコンを指定した場合、順序は上記のリストに従います。このため、特定シンボルに指定したアイコンがある場合、まずこのアイコンが使用されます。指定されたアイコンがない場合、モデル要素のアイコンが使用されます。

アイコン モード

ユーザー指定アイコンで識別されるシンボルには、[アイコン モード] と呼ばれるショートカット メニューがあります。このメニューを選択すると、シンボルが通常のシンボルではなく、可視化されたシンボルになります。

画像ファイル

アイコンは、シンボル、モデル要素、または、適用されたステレオタイプのプロパティで定義されており、そのエンティティの [プロパティの編集] ダイアログを使用して変更できます。このダイアログの [フィルタ] ドロップダウン メニューで [アイコン] を選択して、[Icon File] テキスト フィールドを表示します。このフィールドに入力するテキストは、モデル ファイル（.u2）から画像ファイルのある場所への相対パスです。

アイコンに指定できる画像ファイルのフォーマットは以下のとおりです。

- ビットマップ（ファイル拡張子 .bmp）
- JPEG 圧縮画像（ファイル拡張子 .jpeg または .jpg）
- 拡張メタファイル（ファイル拡張子 .emf）
- GIF（ファイル拡張子 .gif）
- TIFF（ファイル拡張子 .tif、.tiff）
- Targa（ファイル拡張子 .tga、.targa）
- PCX（ファイル拡張子 “.pcx）

注記

アイコン画像に白または透明の背景を使用するとダイアグラムの印刷時に背景が黒くなることがあります。

イメージ セレクタ

ダイアグラム内のシンボルは、イメージ セレクタを使用して ユーザー定義の画像として表示できます。[ImageSelector] アドインをアクティブにすると、[イメージのロード] と [イメージの削除] コマンドが [ツール] メニューに追加され、使用可能になります。

元に戻す

複数レベルの元に戻す操作とやり直し操作を行うことができます。ツール全体（ワークスペース ウィンドウとエディタ）で一般的な元に戻すスタック機能を利用できます。操作が元に戻されると、この操作はまずやり直しスタックに置かれ、元に戻された操作をやり直すことができるようにします。

注記

元に戻す操作は、現在表示されていないダイアグラムにも実行できます。

テキスト編集モードで元に戻す操作とやり直し操作を行う場合、いくつか考慮しなければならない点があります。元に戻す操作は更新ごとに行えます。更新の際、以下のスキームに従います。一連の文字を追加しても、Back Space キー、Delete キー、矢印キーやマウスを使って選択しない限り、更新されません。同様に Delete キーを連続して押しても、他の操作を行わなければ更新されません。

プロジェクトのファイル／リソースを明示的にアンロード（復帰を含む）すると、元に戻すスタックが空になります。

ファイル システムの操作を元に戻すことはできません。

保存を実行しても、元に戻すスタックが空になりません。

モデル参照

モデル定義とその使用の参照を検索するには、類似の機能を持つ一連のショートカット コマンドで行います。これらのコマンドには、コマンドが適用される要素に依存する、コンテキスト依存があります。

参照の一覧表示

［参照の一覧表示］は、［モデル ビュー］のショートカット コマンドで、すべてのモデル要素に適用されます。このコマンドはダイアログを呼び出して、出力ウィンドウの［参照］タブに参照リストを返します。ここでは以下の設定ができます。

- **... へ行われた参照**
  モデル要素のすべての参照先リストです。たとえば、特定クラスを属性タイプとして使用する方法を確認する場合はクラスを参照します。
- **... から行われた参照**
  セクションからの参照リストです。たとえば、属性からタイプとして利用されているクラスを参照します。
- **内包する階層をレポートに含める**
  このオプションを選択すると、選択した要素に含まれる要素へまたは要素からのあらゆる参照が再帰的に検索されます。たとえば、パッケージ外部の定義、パッケージで使用された定義、およびパッケージに含まれた定義がすべて検索されます。

- **内部参照をレポートに含める**
  このオプションを選択すると、オブジェクトまたは内包する自身への階層からの参照、または内包する階層への参照がレポートされます。たとえば、パッケージ内で行われた参照は検索せずに、パッケージとパッケージのコンテンツの利用法が検索されます。

プレゼンテーションの一覧表示

［プレゼンテーションの一覧表示］は、［モデル ビュー］のショートカット コマンドで、すべてのモデル要素に適用されます。すべてのプレゼンテーション要素のリストを出力ウィンドウの［プレゼンテーション］タブに表示します。

既存要素を参照

これは新しいシンボルを配置するショートカット コマンドです。

ナビゲート

［モデル ビュー］のショートカット コマンドで［モデル ナビゲータ］を表示します。、既存のプレゼンテーションがない場合は、［プレゼンテーションの作成］ダイアログを表示します。

［モデル ビュー］で複数のノードを選択した場合、［参照の一覧表示］コマンドと［プレゼンテーションの一覧表示］コマンドは選択したすべての要素に適用されます。

参照

第 i 章「ダイアグラムの操作」の 106 ページ、「シンボルの追加」

ネストされたシンボル

他のシンボル内に配置できるシンボルもあります。他のシンボル内にシンボルを作成すると、作成のコンテキストとして親シンボルのモデル要素が使用されます。

親シンボルに自動サイズ調整が設定されている場合、作成したシンボルに合わせて親シンボルのサイズが調整されます。設定されていない場合は、親シンボルの境界線内に納まるように新しいシンボルのサイズが調整されます。ネストされたシンボルは、親シンボルの境界線の外にドラッグすることはできません。

区画をもつシンボル

区画をもつシンボルには、その区画と区画に含まれるテキストフィールドに関連する特殊な機能があります。

区画にはテキストフィールドが含まれます。そのテキストフィールドはモデル要素と関連付けられます。区画のテキストフィールドは左揃えです。

クラスシンボルのような特定のシンボルは、デフォルトの区画セットを使って作成できます。たとえば、クラスシンボルに、属性と操作の区画を持たせられます。

区画は選択可能です。また区画上で実行できる一連の操作があります。

マウスでカーソルを区画上に移動すると、ツールチップが表示されて区画のタイプを示します。

サイズ変更

区画付きのシンボルのサイズを変更すると、シンボルの大きさに適合しない区画は表示されなくなります。サイズによっては区画全体ではなく区画の一部が非表示になります。

シンボルを区画をすべて表示できる大きさ以上に大きくすると、余白分は均等に各区画に割り当てられます。

区画の作成

区画を持つことができるシンボルには、シンボルのショートカットメニューに［区画］メニューがあります。このメニューのサブメニューには、区画を作成するための一連の操作があります。これらの操作の 1 つを実行すると、区画が作成されます。新しく作成した区画はシンボルの下部に追加されます。

区画の削除

区画は、区画を選択して通常の［削除］コマンドで削除できます。一部の区画はその区画が表示しているモデル要素と直接的に関連付けられます。この場合は、［モデルの削除］コマンドを使って削除することもできます。

区画の移動

区画の順番は、［移動］ツールバーの［上に移動］、［下に移動］コマンドを使って変更できます。

区画上での表示 / 非表示

特定の区画を選択した場合、ショートカットメニューに区画の用途である要素タイプの表示、非表示を選択するメニューが現れます。表示、非表示の操作はそれぞれ適用できる場合にのみ表示されます。

シンボルを選択すると、任意の既存の区画について要素の表示、非表示を選択するメニュー、または特定のタイプのモデル要素用の新しい区画を作成するメニューが表示されます。これらの操作は適用可能な場合にのみ表示されます。複数の区画が同じタイプのモデル要素を表示している場合、表示、非表示の操作は初めの要素に対してのみ実行できます。特定の区画で要素を表示、非表示するには、その区画でショートカットメニューを表示してください。

作成済みの要素をダイアグラムにドラッグした場合、デフォルトではその要素が所有するモデル要素は表示されないことに注意してください。

要素を可視状態にするには、その要素を区画やシンボルにドラッグアンドドロップするか、手動でタイプインします。

ヒント

名前の完成を使用すると、新たなフィーチャを誤って作成することを回避できます。名前をタイプインして、Ctrl+ スペースキー、または Shift+ スペースキーを押すと、複数の候補がある場合はリスト表示されます。

区画のテキストフィールド

要素の削除

区画内のテキストフィールドは、あるモデル要素と関連付けられた別個の表現要素です。このため、モデル要素と結びついたテキストフィールドについては、区画内のテキストを削除できません。ラベルと関連付けられている要素を削除するには、テキストモードに入り、ショートカットメニューから［削除 < 要素 >］コマンドを使用します。

注記

モデル要素と結びついたテキストフィールドを、区画内のテキストの削除によって削除することはできません。削除されるのはその場所にある文字だけです。その行はモデル要素と結びついたままです。

燃える要素と結びついていないテキストフィールドはテキストを削除キーや後退キーで削除することで削除できます。空のテキストフィールドは、カーソルを先頭において後退キーを押すか、末尾において削除キーを押すと削除できます。

要素を非表示

区画のテキストフィールドに表示されている特定の要素を非表示にするには、テキスト編集モードに入って、ショートカットメニューから［非表示 < 要素 >］操作を実行します。

テキストフィールドの移動

テキストフィールドは、［移動］ツールバーの［上に移動］、［下に移動］コマンドを使って上下に移動できます。

共通のライン操作

- ラインのスタイル
- ラインの描画
- 頂点の編集
- ラインの移動
- ラインの削除
- ラインの方向変更と双方向化

ラインのスタイル

ラインに適用できるスタイルには次の 4 種類があります。ラインの作成時に、コンテキストメニューから選択できます。

自動経路選択

ラインは障害物を避けるように自動的に経路を選択します。そのための可能な経路がある場合は直角に曲がります。通常は直線です。

直交

ラインは常に直角で曲がり、頂点とラインセグメントは移動できます。頂点をラインに追加することやラインから削除できます。

非直交

ラインの頂点の移動、追加、削除が自由にできます。

ベジェ

カーブしたラインを使用できます。ラインが選択されると、2 つの制御点が表示されて、カーブを調節できます。

ラインの描画

ツールバーボタンを使用するか、ラインを表現するラインハンドルを使用して、ラインを作成できます。

ラインハンドルを使用してラインを作成する

1. ソースシンボルを選択します。
2. ラインハンドルをクリックします。
3. 頂点を追加します。オプションで終端点をロックできます。
4. ターゲットシンボルまたはラインをクリックします。

ツールバーボタンを使用してラインを作成する

1. ツールバーボタンをクリックします。
2. ソースシンボルをクリックします。
3. 頂点を追加します。オプションで終端点をロックできます。
4. ターゲットシンボルまたはラインをクリックします。

自動経路ラインを除いて、ラインの作成中に頂点を追加できます。頂点の追加ができるときは、カーソルの形が＋マークになります。

ラインの開始点が選択可能な場合は、シンボルエッジ上の開始点の位置を固定できます。ラインをで自動経路選択で作成した場合は、カーソルが「錠前」の形になります。この状態でクリックすると、ラインの開始点は現在の位置に固定されます。自動経路選択以外のラインスタイルで作成した場合は、Shift キーを押しながらクリックすることで開始店を固定できます。

頂点の編集

既存のラインに頂点を追加するには、頂点を作成したいセグメントの上で Ctrl キーを押しながらクリックします。

頂点を削除するには、削除したい頂点上で Ctrl キーを押しながらクリックします。

この操作は直交 または 非直交 のスタイルのラインでのみ可能です。操作が可能な場合は、マウスカーソルの形が変化します。

参照

シンボルの接続

ラインの移動

ラインを移動するには、ラインのいずれかの端点をクリックして、目的の位置までドラッグします。

ラインの削除

ラインは、通常、ダイアグラムから削除してもモデル内に残るモデル要素を表示します。ライン（たとえば関連ライン）を完全に削除したい場合、必ず［モデルからの削除］を実行します。

ラインの方向変更と双方向化

- 描画したラインの方向を変更する場合 （可能な場合）、ラインをクリックして、ショートカット メニューから［逆向き］を選択します。
- ラインを双方向にする場合（可能な場合）、方向やシグナル リストが表示されない端点に近いライン上でラインを右クリックします。ショートカット メニューから［この方向を有効にする］を選択します。
- 双方向のラインでどちらか一方向のみ有効にしたい場合、無効にする端点に近いライン上にカーソルを合わせます。ショートカット メニューから［この方向を有効にする］の選択を解除します。

  この操作の前後で、ラインの方向を目的の方向に変更することもできます。

4

UML 言語ガイド

この章では、DOORS Analyst 4.2 で実装、サポートしている UML 言語について説明します。

- サポートする UML のバージョンについては、UML のバージョンを参照してください。

参照

モデルの操作

ダイアグラムの操作

概要

UML は、ソフトウェアやシステムの仕様決定、可視化、文書化、ビルドなどに使用できるモデリング言語です。以下のセクションでは、さまざまな抽象化レベルでのシステムの構造や振る舞いを表現するために使用できる、各種ダイアグラムや構成要素について説明します。構成要素には、要求や分析など開発の初期フェーズで有用なものと、設計、実装、テストなど開発の後期フェーズで役立つものがあります。このように、さまざまな開発フェーズを連結してシステムを記述できる能力は、UML の大きな長所の 1 つです。

UML のバージョン

DOORS Analyst で使用されている言語は、最新の OMG UML 2.1 Superstructure サブミッションです。DOORS Analyst の実装が言語仕様と異なっている場合もあります。これは、主としてツールの最適化、もしくは、サブミッションの旧バージョンに基づいた設計があるためです。

また、DOORS Analyst では、UML に対して定義されたグラフィック表記とテキスト構文を併用できる点など、言語を一部拡張しています。

ダイアグラム

UML は、システムに対するさまざまな視点を表現するために使う一連のダイアグラムで構成されています。システムの構造に重点を置いたダイアグラムもあれば、エンティティ間の相互作用や、特定条件下で実行される一連のアクションなど、システムの振る舞いの表現のみに使用されるダイアグラムもあります。通常これらのダイアグラムは、システムの仕様を定める主要な手段となります。

DOORS Analyst でサポートされるダイアグラムを以下に示します。

ダイアグラム	目的
ユース ケース図	一連のアクターがユース ケースの観点から、どのように相互作用するのかを表します。通常はサブジェクト、つまり記述されているシステムの文脈で考えます。
シーケンス図	ユース ケースまたは操作のイベント シーケンスを表します。
パッケージ図	パッケージとパッケージ間の依存関係を表します。
クラス図	クラスとクラス間の関係を、通常はパッケージその他の（コンテナ）クラスのスコープで宣言するダイアグラムです。
合成構造図	（コンテナ）クラスの各パート間の接続関係を表し、コンテナの内部構造を示すダイアグラムです。

ダイアグラム	目的
アクティビティ図	並列と相互に関わり合う振る舞いを表現します。複雑な構造を単純化して表示でき、制御の特定のフローに焦点を当てることができます。
相互作用概観図	並列な振る舞いを記述します。ユース ケースを記述するためによく使用されます。
コンポーネント図	コンポーネントの設計に焦点を当て、コンポーネント間の関係と構造を表現します。
配置図	物理的な実装の構造とソフトウェアとハードウェアの関係を表現します。
状態機械図	クラス、状態機械、操作の振る舞いを記述します。

モデルとダイアグラム

モデルとは物理システムを表現したもので、通常は、1 つまたは複数のパッケージに含まれたエンティティによって定義されます。

モデルはシステムの表現であるため、その表現を詳細化することが必要になる場合もあります。たとえば、自動アプリケーション生成の情報源として使用する場合は、要求の可視化として使用する場合よりも、アルゴリズム レベルで詳細に表現する必要があります。

モデルは、ダイアグラムやモデル要素など、システムを表すのに必要なすべてのエンティティを含みます。モデル、厳密に言えばモデルに含まれるモデル要素は、通常、さまざまなダイアグラムでシンボル（モデル要素に対し、プレゼンテーション要素と呼ぶこともあります）を使用して表示されます。

モデル要素

モデルの主な内容は、クラス、属性、操作、アクション、制約などのモデル要素です。モデル要素は、エンティティのすべての特性を格納するために使用します。この仕組みで、ダイアグラムのモデル要素はさまざまな側面を示すことができます。たとえば、1 つのクラス図でクラスの属性や操作を示し、別のクラス図で、定義されているクラス階層を示すことができます。これらのダイアグラムでは、同じモデル要素の異なる側面を表示していることになります。

シンボル

シンボルは、モデル要素（のパート）を描画された図形として可視化するものです。各シンボルは、ダイアグラムに示される二次元オブジェクトです。シンボルは、サイズと、ダイアグラムの座標系における位置を示すようになっています。

クラス シンボルなど、ほとんどのシンボルは、対応するモデル要素を直接的に可視化したものですが、テキスト シンボルなどのように、基礎となるモデル要素がないものもあります。シンボルは、特定のダイアグラムのみに関連付けられます。

モデル要素とシンボルの区別は重要ですが、日常の言葉では、2 つの区別は曖昧になる場合が多いと思います。クラス モデル要素やクラス シンボルは、多くの場合、単にクラスと呼ばれます。

モデル要素のさまざまなビュー

124 ページの図 1 は、3 つの異なるビューを使用して表した、モデル要素 a の例です。1 つ目のビューでは、このモデル要素はクラス C の属性として表されています。2 つ目では、クラス C とクラス D の関連の端として示されています。3 つ目では、クラス C の内部構造の一部として示されています。

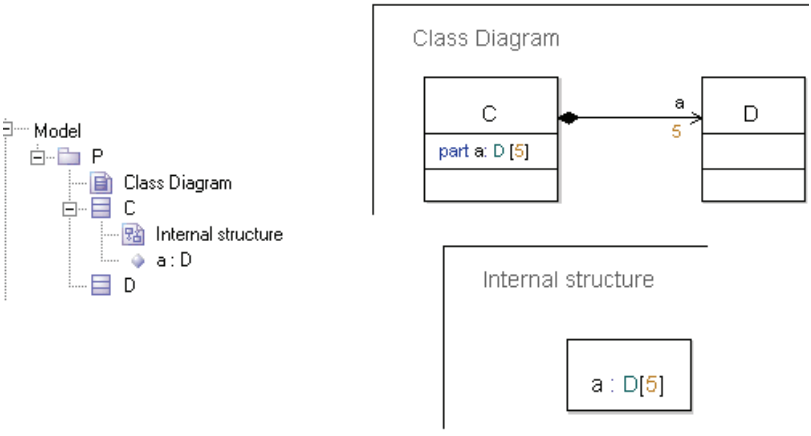

図 1: 属性のさまざまなビューの例

また、「モデルの削除」（モデルからの削除）と「ダイアグラムからの削除」の違いも重要です。左側のブラウザ ビューにはモデルが示され、ブラウザ ビューから要素を削除すると、要素は、モデルからだけでなく、モデルが表示されているダイアグラムからも削除されます。右側の 2 つのダイアグラム ビューでは、同じ属性 a が 3 通りの方法で表示されています。1 つ目はクラス C の属性入力領域に、2 つ目はクラス C とクラス D の関連の端として、3 つ目はクラス C の内部構造の一部として表示されています。

シンボルとモデル要素の削除

要素をダイアグラムから削除する方法は 2 つあります。通常の削除では、シンボルは削除されますが、モデル要素はモデルに残ります。モデルからの削除では、要素はモデルからだけでなく、モデルが表示されている他のすべてのダイアグラムからも削除されます。

一部のダイアグラムでは、モデル要素とシンボルが密接に接続されています。このようなダイアグラムには、シーケンス図、状態機械図があります。これらのダイアグラムでは、シンボルとモデル要素は 1 対 1 のマッピング関係にあり、一方が削除されると、もう一方も削除されます。（つまり、これらのダイアグラムでの削除はモデルからの削除と同じです。）これは、たとえば、アクションや遷移に当てはまりますが、ステートには当てはまりません。

参照

シンボルの追加

シンボルの移動

シンボルのサイズ変更

シンボルの接続

シンボルのテキスト フィールドの編集

シンボルのコピー、切り取り、削除、貼り付け

言語構成要素一覧

次の表に、すべての具象モデル要素、その他 UML の重要な言語構成要素を示します。

UML モデル要素
イベント受信 , タイム イベント受信 , アクセス , アクション（操作本体、状態機械、状態機械図）, アクション （相互作用図とシーケンス図）, アクションノード（アクティビティ図）, アクティブ クラス , アクティビティ , アクティビティ終了 , アクター , 集約 , 任意値（any）式 , アーティファクト , 代入 , 関連 , 属性
振る舞いポート
選択 , クラス , 分類子 , コメント , コンポーネント , 合成状態 , 合成 , 複合文 , 条件式 , 定数 , コネクタ（合成構造図）, コネクタ（アクティビティ図）, 継続 , 共通リージョン , 生成
データ型 , 分岐（状態機械図）, 分岐（アクティビティ図）, 依存 , デプロイメント , デプロイメント スペシフィケーション , ダイアグラム , 消滅
エントリ接続ポイント , 実行環境 , 終了接続ポイント , 式 , 拡張
フィールド式 , フロー終了 , フォーク
汎化 , ガード
履歴の次のステート
命令式 , 実現化 , インポート , インデックス式 , 開始ノード , インライン フレーム , 相互作用 , 相互作用参照 , インターフェイス , インターナル
ジョイン , ジャンクション
ライフライン（生存線）リテラル
表現 , マージ , メッセージ , メソッド , メソッド呼び出し

UML モデル要素
New, 次のステート , ノード , now 式
オブジェクト ノード , offspring, 操作 , 操作本体 , シグナル送信アクション（出力）
パッケージ , Parent, パート , アクティビティ区画 , Pid 式 , ピン , ポート , 定義済み , プロファイル
範囲チェック式 , 実現化インターフェイス , 要求インターフェイス , リターン
保存 , Self, シグナル送信 , Sender, シグナル , シグナル リスト , シグニチャ , 開始遷移 , ステート , 状態機械 , 状態機械実装 , ステート式 , ステレオタイプ , 停止 , サブジェクト , シンタイプ
タグ定義 , タグ付き値 , ターゲット コード式 , アクション（タスク） , this 式 , タイマー, タイマー アクティブ式 , タイマー リセット , タイマー リセット アクション , タイマー設定 , タイマー設定アクション , タイマー タイムアウト , 遷移
ユース ケース

スコープ、モデル要素、ダイアグラム

パッケージやクラスなど、一部のモデル要素は名前スコープを表現できます。つまり、これらのモデル要素には、他のモデル要素の定義を含めることができます。名前スコープ内のすべての定義には、一意の名前を付ける必要があります。一意の名前を付けないと、セマンティック チェッカーから注意を促されます。スコープは、グループを構成するモデル要素の「コンテナ」または「グルーピング」と考えるとよいでしょう。

ほとんどのスコープには、モデル要素だけでなく、モデル要素が表示されているダイアグラムも含まれます。次の表に、各スコープに使用できるダイアグラムを示します。

スコープ単位	使用できるモデル要素	ダイアグラム
パッケージ	パッケージ , クラス , ユース ケース , アーティファクト , ステレオタイプ , 関連 , データ型 , インターフェイス , シンタイプ , 選択 , 操作 , 属性 , シグナル , シグナル リスト , タイマー , 状態機械	クラス図 シーケンス図 ユース ケース図
クラス	クラス , アーティファクト , ステレオタイプ , データ型 , インターフェイス , シンタイプ , 選択 , シグナル , シグナル リスト , タイマー , 属性 , 操作 , ユース ケース , 状態機械 ,	クラス図 合成構造図
ユース ケース	相互作用 , 状態機械実装 , 操作本体	シーケンス図 状態機械図

スコープ単位	使用できるモデル要素	ダイアグラム
相互作用	ライフライン（生存線）	シーケンス図 ユース ケース図
ステレオタイプ	属性	
データ型	リテラル , 操作	
選択	属性 , 操作 ,	
インターフェイス	シグナル , タイマー , 属性 , 操作	
操作	操作本体 , 状態機械実装 , 相互作用	
操作本体	状態機械 , クラス , アーティファクト , ステレオタイプ , データ型 , インターフェイス , シンタイプ , シグナル , シグナル リスト , タイマー , 操作 , 属性	状態機械図
状態機械実装	クラス , アーティファクト , ステレオタイプ , データ型 , インターフェイス , シンタイプ , シグナル , シグナル リスト , タイマー , 操作 , ステート , アクション , 属性	状態機械図 クラス図 ユース ケース図
複合文	アクション , 属性	

オーバーロードされた定義

特定の種類の定義は、スコープ内で同一の名前を複数持つことができます。これは、操作、シグナル、タイマー および 状態機械などの振る舞い特性にあてはまります。これらの定義は、名前だけではなくパラメータの型も使って識別されます。操作の名称とパラメータ型の一覧を、振る舞い特性の「シグニチャ」と呼びます。同じスコープ内のすべての振る舞い特性は、一意のシグニチャをもつ必要があります。同一スコープ内で同じ名前を持ち、シグニチャが異なる 2 つの振る舞い特性は、「オーバーロードされている」といいます。

一般的な言語構成要素

UML には、複数のダイアグラムに共通する言語構成要素がいくつかあります。

名前

すべての UML モデル要素には、名前、すなわち識別子があります。名前には、一定のルールがあります。

命名ルール

名前には、文字、数字、_（アンダースコア）が使用できます。

名前の先頭は数字でなく、文字またはアンダースコアでなければなりません。また、特殊な場合として、先頭が常に ~（チルダ）になるデストラクタ名もあります。

識別子でのスペースと特殊文字の使用

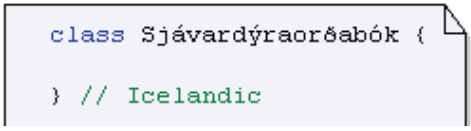

図 2: 識別子での特殊文字の使用

名前を単一引用符で囲めば、上記の制限を取り除くことができるので、名前の一部として（ほとんどの）任意の文字を使用できます（128 ページの図 2 を参照）。たとえば、名前を単一引用符で囲めば、名前の一部としてスペースを使用できます（128 ページの例 1 を参照）。

文字列の処理に使用できる、いくつかのエスケープ文字があります。¥n、¥t、¥b、¥r、¥f は、charstring 内（“” 内）に入れるか、文字（'¥n' など）として使用できます。

「¥"」は charstring 内で使用でき、「¥'」は文字として使用できます。「¥¥」は charstring 内または文字列として使用でき、円記号を表します。単一引用符で囲まれたその他のエスケープ文字（'¥+'、'¥s'）は、識別子（+、s）と解釈されます。二重引用符で囲まれている文字列内で円記号に続く文字は、その文字自体を表します（“a¥qa” は” aqa” を表す）。

```
¥n: new line
¥t: tab
¥b: backspace
¥r: carriage return
¥f: form feed
¥": quotation mark, e.g. "my ¥"quoted¥" word"
¥': apostrophe character, '¥''
¥¥: yen mark
```

例 1: 識別子でのスペースの使用

```
Boolean 'has finished'=false;
```

大文字と小文字の区別

識別子では、大文字と小文字が区別されます。つまり、違いが大文字か小文字かだけである名前も、異なるものとして認識されます。

例 2: 大文字と小文字の区別

```
Integer MyInt, myint; // Two distinct attributes
```

参照

名前付きの定義は、モデル内の他の場所から参照されることがあります。単純なケースとして、参照が定義の名前から構成される場合があります (必要に応じて、単一引用符で囲みます)。通常は、参照はさらに複雑になります。

- 参照は修飾子を含むことがあります。

スコープをまたがって同じ名前を持つ定義を区別するためには、名前に修飾子をつけます。修飾子とは、スコープパスと特殊なスコープ解決演算子 「::」から成る識別子の接頭辞です。グローバル名はパスがありませんので、「::」が先頭に置かれます。「::」で始まる修飾子を絶対修飾子と呼び、その他のものを相対修飾子と呼びます。

- 参照は実テンプレート引数を含むことがあります。

参照定義がテンプレートである場合 (つまり、参照定義が テンプレート パラメータ をもつ場合) は、参照は、テンプレートパラメータの実の値を含む必要があります。実テンプレート引数は、「<」「>」ブラケット内の名前に続くカンマ区切りのリストで与えます。

- 参照はパラメータ型のリストを含むことがあります。

振る舞い特性を参照する場合は、パラメータ型の名前を追加する必要があります。これは、名前とともにパラメータの型が振る舞い特性のシグニチャの一部となるからです。パラメータ型の名前は、名前の後の括弧で囲んで与えます。

例 3: 異なる種類の参照

この例では、2 つの属性が修飾名を使って型を参照しています。

```
::Predefined::Integer i;

UtilityTypes::Sorts::ClientIdx j;
```

型がテンプレートクラスの場合は、実テンプレート引数を指定する必要があります。

```
MyClass<Integer, 4> k;
```

操作のような振る舞い特性を参照する際は、パラメータ型を指定する必要があります。このような参照の場合は、通常の操作の呼び出しと区別して文法的な曖昧さを排除するために、キーワード「operation」を使っていることにも注意してください。

```
OperationReference r = operation foo(Integer, Boolean);
```

予約語

DOORS Analyst に予約されており、モデル要素の名前として直接使用できない名前があります。名前の一覧は「UML 2 テキスト構文」を参照してください。

これらの単語を単一引用符で囲んで定義の名前に使用できますが、混乱の原因になるため、どうしても必要な場合以外は、使用しないでください。

例 4: 予約語を単一引用符で囲んだ使用方法

```
Integer 'class'; // confusing attribute name, but valid
```

代替構文

UML で定義されているグラフィック表記に加えて、通常の「テキスト」でモデルを記述するための補足テキスト構文が定義されています。この記法は、グラフィック シンボルの代わりに使用することも、またはグラフィック シンボルと併用することもできます。

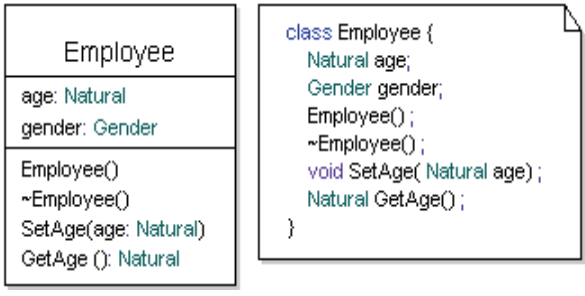

図 3: クラス シンボルの構文とテキスト構文の違いの例

130 ページの図 3 では、同じモデル要素が、1 つのダイアグラム中で 2 度表示されています。左側ではグラフィック表記が使用され、右側ではテキスト シンボル中にテキスト構文が使用されています。これらのビューのいずれかに変更を加えると、自動的にもう一方に反映されます。

共通の要素プロパティ

以下のプロパティは、さまざまなモデル要素に使用されます。これらのプロパティは、プロパティ エディタで表示して制御できます。

Visibility（可視性）

多くのモデル要素には可視性があります。可視性を使用して、要素に対する、要素が定義されているスコープ外からのアクセス権限を決定します。スコープ内では、可視性に関わらず、すべての要素にアクセスできます。可視性には、以下に示す複数のレベルがあります。

- **Public**
  要素が public 可視性を持っている場合、そのコンテナを見ること（アクセス）が可能なすべての要素が、その要素にもアクセスできます。
- **Protected**
  要素が protected 可視性を持っている場合、そのコンテナのサブクラス、および要素と同じスコープ内のすべての要素が、その要素にアクセスできます。
- **Private**
  要素が private 可視性を持っている場合、同じスコープ内の要素のみが、その要素にアクセスできます。
- **Package**
  要素が package 可視性を持っている場合、同じパッケージ内のすべての要素がその要素にアクセスできます。
- **なし**
  可視性が指定されていない場合は、要素は以下の表に従ってデフォルトの可視性が設定されます。

定義のデフォルトの可視性は、スコープとタイプによって異なります。

スコープ	可視性
Class, Choice, Stereotype, Collaboration, Artifact	Private
Package	Public
Interface	Public
DataType	Public

注記

リテラルは必ず public 可視性を持ちます。
データ型のすべてのリテラルと public 静的メンバーは、修飾子のないデータ型の外側でも見えます。修飾子は曖昧性を解決するためにのみ必要です。たとえば、同一スコープ内の 2 つのデータ型が同じ名前のリテラルを持つ場合などです。

Virtuality（仮想性）

仮想性は、クラスなど分類子を汎化し、特化クラスの内部モデル要素が再定義可能かどうかを決定する場合に必要になります。

仮想性は、タイプ内の要素（特化可能な分類子）にのみ適用されます。コンテナが特化されている場合、各内部要素の個々の仮想性によって、その要素が変更可能かどうかが決まります。

- **Virtual**
  内部要素が仮想（Virtual）の場合、コンテナを特化して要素を再定義（変更）できます。

- **Redefined**
  特化コンテナ内の要素の仮想性が再定義（Redefined）の場合、基底コンテナの元の要素の定義は変更されています。基底コンテナの元の要素は仮想でなければなりません。
  再定義された要素もまだ仮想です。すなわち、コンテナが再度特化されれば、要素をさらに再定義（変更）できます。
- **Finalized**
  特化コンテナ内の要素の仮想性が最終（Finalized）の場合、基底コンテナの元の要素の定義は変更されています。基底コンテナの元の要素は仮想でなければなりません。最終であるということは、コンテナが再度特化されても、この要素をさらに再定義することは不可能であるということです。つまり、仮想性が最終であるということは、再定義されているがもはや仮想はでないという意味になります。
- なし
  内部要素に仮想性がない場合、コンテナが特化されると、要素を再定義（変更）できません。

Derived（導出）

要素が導出されている場合は、他の要素を使用することでその値を計算できます。計算法の指定方法はコンテキストによって決まります。

導出要素の典型的な使用法は、導出属性です。属性にアクセスする際に使用する導出規則を、アクセス演算子「get」や「set」を使用して指定できます。

例 5: S 導出属性のための導出規則の指定

```
Integer y;
Integer / x
  get { return 5; }
  set { y = value; };
```

他のプロパティ

- **External**
  定義が外部であるとは、定義がこのモデルの外部にあることを意味します。付属のコード ジェネレータは、外部要素に対してはコードを生成しません。したがって、外部要素は、外部で利用可能な定義のモデル表現と見なすことができます。
- **Abstract**
  分類子が抽象である場合、この分類子を直接インスタンス化できません。通常は、この抽象分類子は別の分類子によって特化されています。その場合は、特化分類子をインスタンス化できます。
- **Static**
  定義が静的であれば、包含する分類子のすべてのインスタンスがこの要素の実装を共有します。つまり、同じデータを使用します。したがって、静的な定義は、定義されている分類子のインスタンスがなくても使用できます。

パラメータ

操作、シグナル、状態機械などの振る舞い特性を表す定義は、パラメータを持つことができます。一般的な形式（分類子シンボルとプロパティ エディタで使用される）は次のとおりです。

```
name:type, name2: type2
```

パラメータにより、呼び出しから振る舞いへとデータが流れる（フローする）方向を指定できます。

- **In**（デフォルト）
  データは、呼び出し側から呼び出される振る舞いに渡されます。
- **In/Out**
  データは、呼び出し側から呼び出される振る舞いに渡され、さらに、呼び出された振る舞いから呼び出し側へ戻されます。
- **Out**
  データは、呼び出された振る舞いから呼び出し側へ戻されます。
- **Return**
  データは、呼び出した振る舞いから呼び出し側に、呼び出しの結果の戻り値として渡されます。戻り値パラメータとしては 1 つのみが許されます。

テンプレート パラメータ

テンプレート パラメータとは、使用する文脈に依存しない定義を行って、より動的で柔軟な方法で分類子を使用するための概念です。テンプレート パラメータは、**コンテキスト パラメータ**とも呼ばれます。

クラスや操作など、特化やインスタンス化が可能な要素には、テンプレート パラメータを持たせることができます。

テンプレート パラメータは、インスタンス化、または、そのテンプレートパラメータをもつ分類子が特化や再定義される際に、実際のパラメータ「値」と結び付けられます。特化の際には、テンプレート パラメータのサブセットを値と結び付けることもできます。インスタンス化の際は、すべてのテンプレート パラメータを値と結び付ける必要があります。

原則として、テンプレート定義が参照される場合にはテンプレートパラメータについて実際の値が指定されている必要があります。ただし、以下の 2 つの例外があります。

1. テンプレートパラメータがデフォルト値をもつ場合、実際の値を与える必要はありません。この場合はデフォルト値が使用されます。
2. テンプレートパラメータを使用した振る舞い特性の呼び出しにおいて、呼び出しで使用されている実際の呼び出し引数でそのテンプレートパラメータを引数としていない場合は、実際の値を指定する必要はありません。

演算子 `reinterpret_cast<T>` と `cast<T>` は、実際のテンプレート パラメータとしては使用できません。`reinterpret_cast<T>` または `cast<T>` 演算子を含むインスタンス化テンプレートは、名前解決では解決できません。

例 6: キャスト演算子を含むインスタンス化テンプレート

この制限を以下の例に示します。

```
template<const Integer x>
class MyTemplate { }
enum E { L }

/* These template instantations cannot be resolved */
MyTemplate<cast<Integer>(L)> myVar1;
MyTemplate<reinterpret_cast<Integer>(L)> myVar1;
```

定義済みの名前

付属のユーティリティ パッケージ Predefined には、役に立つデータ型、リテラル値、演算があらかじめ定義されています。このエンティティの名前は特に予約されてはいませんが、誤解による予期せぬ障害を招く可能性があるため、この名前を他のエンティティには使用しないことを推奨します。

参照

定義済み

ユース ケース モデリング

ユース ケース モデリングでは、主として、システムやそのシステムの一部分の「使われ方」を、各要素の振る舞いに対する要求に基づき、また、相互作用対象のアクターに関連付けて決定することに焦点を当てます。

ユース ケース図

ユース ケース図は、ユース ケースとアクターの関係を示すことにより、システムの使用の状況を説明するものです。ユース ケース図により、システムの動的側面が静的に表示されます。

例

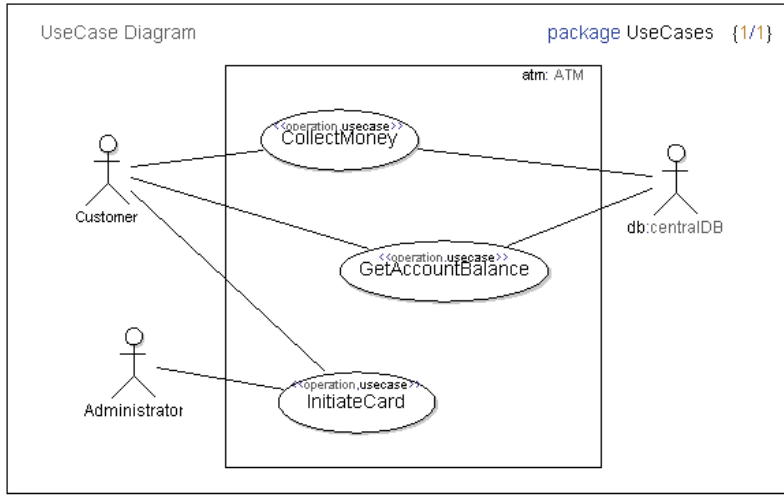

図 4: アクター、ユース ケース、サブジェクト、アクターとユース ケースの関連を示したユース ケース図

ユース ケース図のモデル要素

ユース ケース図には、以下の要素が使用されます。

- ユース ケース
- アクター
- サブジェクト
- 依存
- インクルード
- 拡張
- 汎化
- 関連

ユース ケース図の作成

ユース ケース図は、パッケージ、クラス、コラボレーションに含めることができます。

1. ［モデル ビュー］でパッケージ（クラス、コラボレーション）を選択します。
2. ショートカット メニューから［新規］、次に、［ユースケース図］を選択します。

ツールバーを使ってユース ケース図を描画したり、モデルからユース ケースをドラッグしてユース ケース図に入れることもできます。

- ツールバーを使用するには、まずユース ケース シンボルをクリックしてから、ダイアグラム内のユース ケース シンボルの配置場所をクリックします。

ユース ケース

ユース ケースは、システムまたはシステムのパートの機能の、一貫した単位を表します。通常、システムはクラスで表します。機能は、多くの場合、システムの振る舞いなど、システムと 1 つまたは複数の外のアクターとのやりとりに関連して表現されます。

ユース ケースは多くの点で操作と類似しており、ステレオタイプが <<use case>> の操作としてモデリングされます。

シンボル

図 5: ユース ケース シンボル

ユース ケース図では、ユース ケースは、ユース ケース シンボルを使用して可視化されます。ユース ケースは、以下のスコープ内で指定できます。

- パッケージ
- クラス
- コラボレーション

ユース ケースの記述

ユース ケースの振る舞いは、以下を使用して定義します。

- 相互作用
- 状態機械
- 操作本体

テキストを使用してユース ケースを記述できます。この場合、通常は、テキストに一定の構造をもたせることになります。ユース ケース名、その目的、事前条件、事後条件、例外的な場合、ユース ケースが実行するアクションの形で記述された実際の機能などを、整理して記述するためです。

例 7: テキストで記述されたユース ケース

```
Use case:CloseAccount
Goal:Close a user account and make sure the balance of the
account is settled
Preconditions:Customer has an open account
Postconditions:Customer has closed the account and has paid
outstanding dues
Description:
1. Check balance of account
2.a If balance is positive, pay customer
2.b If balance is negative, collect payment from customer
3. Terminate card associated with account
4. Close account
```

ユース ケースの命名

ユース ケースの名前には、述語を使うのが一般的です。つまり通常は、「do something」などのように、「動詞と目的語」を含んだフレーズを使用します。引用符で囲んだ名前を使用すれば、名前に空白を含めることもできます。

例 8: 引用符で囲んだユース ケース名

```
<<usecase>> void 'Open Account' ();
```

動詞と名詞の間に空白を入れない記法もよく使われます。

アクター

アクターは、機能の起動やユース ケースの情報源といった形でユース ケースに関与するエンティティを表現します。

シンボル

図 6: アクター シンボル

アクターは、ユース ケース図で、棒線画を使用して表されます。アクターは、関連を使用してユース ケースと接続されます。

アクターのロール

ユース ケース図では、アクターとユース ケースの関係を示すことに重点が置かれます。アクターは、多くの場合 1 つまたは複数のサブジェクトのコンテキストでユース ケースに関与しているエンティティです。アクターは、ユース ケースの定義対象のサブジェクトの外部にある、ユーザー（人間）、外部ハードウェア デバイス、他のサブジェクトなどの場合があります。アクターは、1 つの物理エンティティとは限りません。たとえば、コンピュータ ネットワーク全体であってもかまいません。

複数の異なるユース ケースでは、同一の物理エンティティを表すために、別のロールをもつ複数の別のアクターを使用できます。また、1 つのアクターで、複数のユース ケースの異なる物理エンティティを示すこともできます。

アクターは、クラスのパートまたはインスタンスを参照します。

繰り返しますが、ユース ケース図では、アクターとユース ケースの関係を示すことに重点が置かれます。しかし、アクターのタイプに注目すると有用な場合もあります。たとえば、継承を使用してアクター間の関係を示したり、アクターのプロパティを示したりすることです。これらの情報は、アクターが <<actor>> ステレオタイプのクラス シンボルとして表示されているクラス図で表示されます。

アクター シンボルは、アクターに適用されたステレオタイプまたはアクターが参照するクラス（アクターにステレオタイプが適用されていない場合）を可視化します。

注記

アクターのステレオタイプ use case、actor、subject は表示されません。

サブジェクト

サブジェクトは、一連のユース ケースのシステム境界を定義します。サブジェクトはシステム、サブシステム、クラスを表します。サブジェクトは、クラスのパートまたはインスタンスを参照します。

サブジェクトは、UML 1.X のユース ケースのシステム境界に該当します。

シンボル

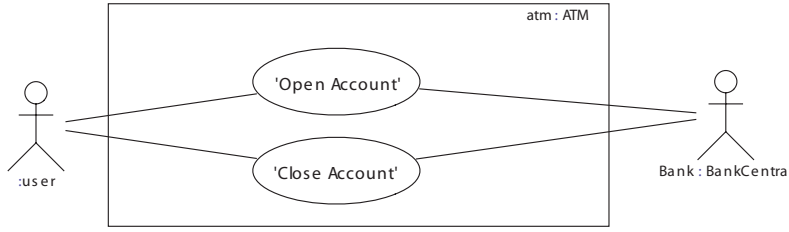

図 7: サブジェクト シンボル（ATM）

ユース ケースを、サブジェクト シンボルの内部に含むことができます。サブジェクト シンボルは、アクティブ クラスなどの振る舞いを表す一連のユース ケースを囲むように描画します。名前とクラス タイプは、サブジェクト シンボルの右上隅に入力できます。

関係

コラボレーションやユース ケース図で以下の関係が使用できます。

関連

関連は、アクターとユース ケースの間で使用され、アクターがそのユース ケースに関与していることを示します。逆に言うと、ユース ケースは、アクターによって実行されていることになります。1 つのアクターが複数のユース ケースに関与したり、1 つのユース ケースに複数のアクターが関与したりすることもできます。

インクルード

インクルード関係は、複数のユース ケース間で使用され、あるユース ケースが別のユース ケースの一部であることを示します。いわば、大きいユース ケースを小さいユース ケースに分割する仕組みです。インクルードする側のユース ケースは、通常それ自身はあまり意味はなく、インクルードされているユース ケースに依存することがほとんどです。

拡張

拡張関係は、複数のユース ケース間で使用され、ユース ケースをいつどのように拡張ユース ケースに挿入するかを示します。拡張ユース ケースは、それ自体で完成していなければなりません。拡張では、通常は、一定の条件下で使用する補足的な機能について記述します。

依存

依存は、ユース ケース間やアクター間で指定されます。依存はエンティティ間の関係の仕方については何も示しません。

2 つのユース ケース間に依存が作成されると、暗黙的にインクルード関係が成立します。

汎化

汎化はユース ケース間で指定できます。つまり、あるユース ケースが、より一般的なユース ケースを特化します。クラスに関連付けられたアクターでは、汎化を指定できます。汎化テキストは非形式的です。

参照

UML の関係

シナリオ モデリング

シナリオ モデリングでは、主としてシステムやサブシステムの用途、用法についてのシナリオを記述することに焦点を当てます。このシナリオは、ライフライン上で生起するイベントのシーケンスとして記述されます。

モデリング作業を通じて、メッセージのやりとりを詳細化してゆくことで、システム内のコンポーネント間での責任分割、さらにシステムと、そのシステムと相互作用する外部アクターの間の境界線がより明確になります。

シナリオ モデリング作業は、多くの場合、分析作業の早期に行われますが、設計作業でも、より厳密な形で行われることがあります。作成されるシナリオは、システムとシステム コンポーネントの動的インターフェイスの仕様です。シナリオには通常、以下の 2 つの目的があります。

- コンポーネントの振る舞いモデリングの基盤となる
- テスト ケースの基盤となる

UML では、シナリオは相互作用を使用してモデリングされ、イベントは、このセクションで説明するシーケンス図に示されます。相互作用概観図は、個々の相互作用の制御と調整に使用されます。

シナリオ モデリングは、多くの場合、ユース ケース分析の一環として行われます。ユース ケースごとに、そのユース ケースに関連する振る舞いを記述する相互作用が生成され、シーケンス図を使用して相互作用が可視化されます。

シーケンス図

説明

シーケンス図は相互作用を記述するものです。シーケンス図によって、ライフラインやイベント間のメッセージのやりとりが可視化されます。

例

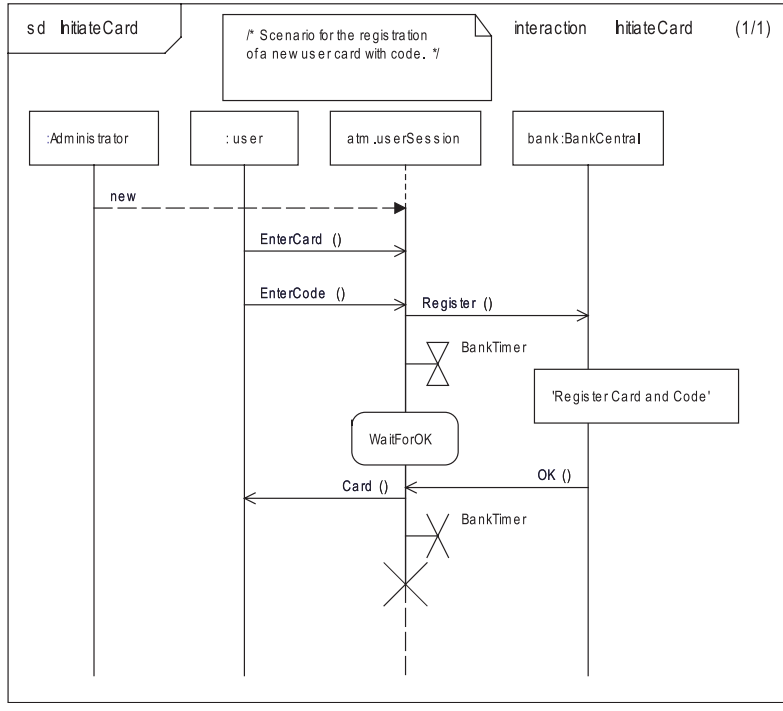

図 8: シーケンス図

シーケンス図のモデル要素

シーケンス図では、以下の要素を使用します。

- ライフライン（生存線）
- メッセージ
- アクション
- ステート

- 相互作用参照
- タイマー イベント
- 生成
- 消滅

シーケンス図の作成

シーケンス図は、相互作用の実装の図形記述です。たとえば、あるパッケージ内でシーケンス図を作成すると、シーケンス図は相互作用とその実装の下に自動的にカプセル化されます。

また、シーケンス図に、操作とユース ケースのような、他の振る舞いの実装を付与できます。この場合、シーケンス図はその振る舞いの内部に作成します。

相互作用

相互作用は、ユース ケース、操作、その他振る舞いを持つエンティティの振る舞いの記述です。相互作用では、パート間の情報交換に重点が置かれます。相互作用は通常シーケンス図によって記述されます。

相互作用の意味は、相互作用から導出される一連のトレースによって定義されます。トレースとは、イベント発生のシーケンスです。このシーケンスは、完全に整列されているとはかぎりません。トレースによるシナリオには、可能なものも不可能なものもあります。

相互作用は、他の相互作用から参照できるので、再利用が可能です。通常は、別のユース ケースを参照する相互作用参照シンボル、または振る舞いの定義として相互作用を含む操作によって行います。ライフライン分解によって相互作用を参照することもできます。

相互作用の通常の使用方法には、以下の 2 つがあります。

- システムとコンポーネントの、外部から見える振る舞いの指定
- システム実行のトレースの記述

参照

シーケンス図

ユース ケース

相互作用参照

相互作用参照は、シーケンス図で相互作用の参照を表すために使用します。参照先の相互作用は、通常、それ自体のシーケンス図で記述されます。相互作用参照で使用する名前は、相互作用自体の名前でなく、相互作用を含むユース ケースまたは操作の名前です。

相互作用参照は、以下の 2 つの点で便利です。

- 重要なメッセージのやりとりに焦点を当てながらも詳細な相互参照を隠す、カプセル化機構として使用できます。
- 相互作用の記述の再利用を可能にします。

シンボル

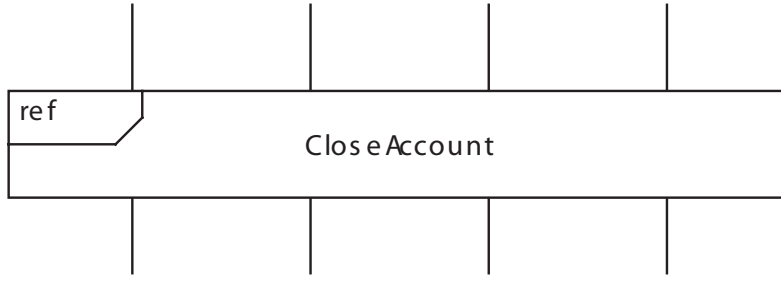

図 9: 相互作用参照

構文

相互作用参照シンボルには、ユース ケース、操作その他振る舞いを持つエンティティを参照する名前が含まれます。

参照

相互作用

ユース ケース

シーケンス図

ライフライン（生存線）

ライフラインは、1 つの相互作用に関与する個々の参加者を表します。パートとストラクチャフィーチャには 1 よりも大きい多重度を持たせることができますが、ライフラインは、1 つの相互作用エンティティのみを表します。ライフラインが、多重度が 1 よりも大きいパートを表す場合は、インデックスにより、特定のインスタンスを選択する必要があります。

シンボル

ライフライン シンボルは、「頭部」と生存線である「軸」から構成されます。ライフラインがまだ生成されていない場合は、軸は破線になります。ライフラインが消滅する（インスタンスが終了する）と、軸が再度破線になります。

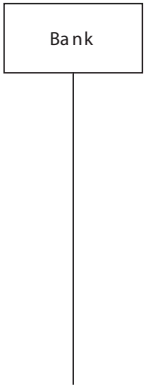

図 10: ライフライン シンボル

ライフラインの作成

ライフラインを生成するには以下の手順を行います。

- ［ダイアグラム要素の作成］ツールバーを使用して［ライフライン］シンボルを選択します。ライフライン シンボルをダイアグラムに配置します。**パート名**または**クラス名**などの適切な情報をヘッダーに入力します。
- クラス シンボルをモデルからシーケンス図にドラッグして、このクラスを表現するライフラインを作成します。ライフラインはクラスのどのインスタンス、ヘッダー内のテキスト**クラス名**で表現します。
- モデルからパート シンボルをドラッグして、このパートを表示するライフラインを作成します。ライフラインはパートのインスタンスを、ヘッダー内のテキスト**パート名**（スコープ修飾子が必要な場合は、**修飾子 :: パート名**）で表現します。

イベントの整列

ライフラインに沿ったイベント発生の順序は、イベントが実際に発生する順序を表す重要なものです。ただし、ライフライン上のイベント発生間の絶対距離は、特に意味をもちません。

イベントの順序は、1 つのライフライン上では厳密に指定されています。しかし、通常は、複数のライフライン上ではイベント間に特定の順序はありません。また、各非同期コンポーネントをそれ自体のライフラインで記述した相互作用やシーケンス図を使用して、分散システムを記述することも可能です。

一般に、複数のライフライン上のイベントの順序を確定するための唯一の仕組みは、メッセージ送信による同期です。この、シーケンス図の順序確定機構は、**部分的整列**（partial ordering）と呼ばれます。なぜならば、完全な順序の確定ではなく、また、まったく不規則な順序付けでもないからです。

非同期でも分散型でもないシステム（スレッドのない通常のプログラム）の場合は、当然ながら、非同期のケースよりも厳密に順序を解釈できます。

ライフライン分解

ライフラインで、合成状態、つまりパートのあるオブジェクトを参照できます。この方法によって、相互作用の複雑性は軽減され、最も重要なメッセージのやりとりに焦点を当てられます。

ただし、場合によっては、内部のやりとり、つまり合成 オブジェクトのパート間の詳細なメッセージのやりとりも確認するとよい場合があります。ライフラインの分解機構によって、同じ振る舞いに対して、ハイレベルの概要と詳細の 2 種類の情報を記述できます。詳細な相互作用は、ライフラインの頭部から参照されて、別のユース ケースまたは操作で定義されます。146 ページの図 11 の例を参照してください。

分解の例

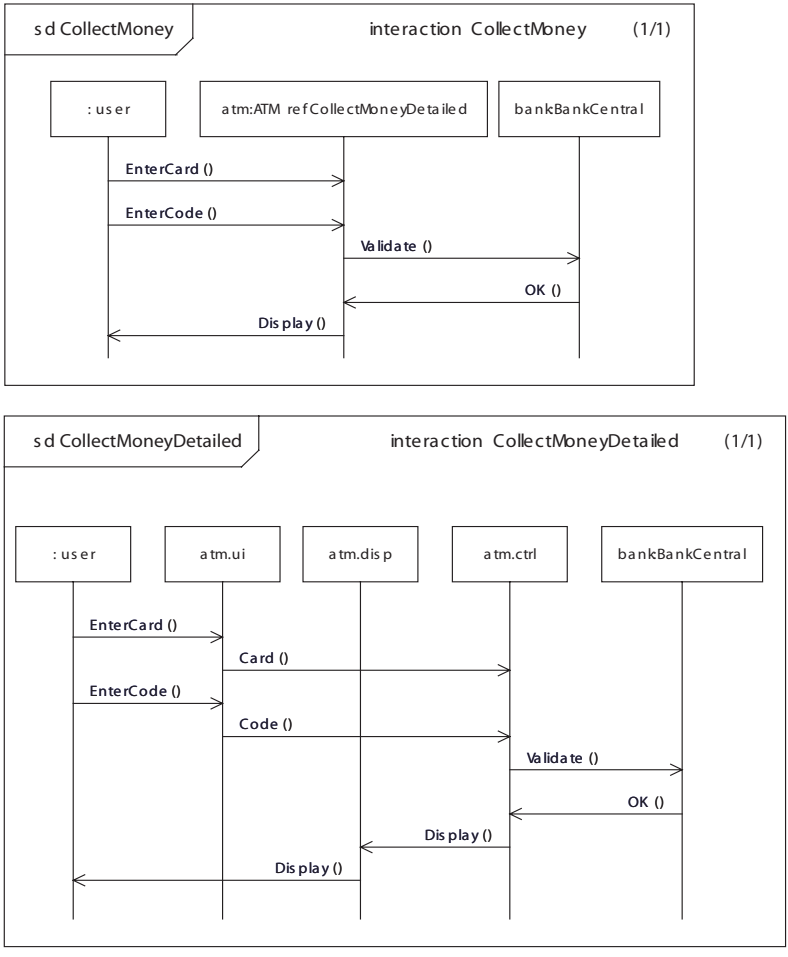

図 11: ライフライン分解の例

構文

ライフラインでは、次の構文が使用できます。

`Bank`
パート、ポート、属性、またはサブジェクトを参照するインスタンス名

`Bank:BankCentral`
インスタンス名と、クラスを示すタイプ名

`:BankCentral`
クラスを示すタイプ名

`atm[3]`
多重度を 1 インスタンスに減少させる セレクタ式を持つインスタンス名

`atm.Display`
パートを参照する属性を持つインスタンス名

`atm ref OpenAccountDetailed`
インスタンス名と、別の相互作用とシーケンス図で記述されるユース ケースまたは操作を参照するライフライン分解

`atm[2].Display:ATM ref CloseAccountDetailed`
セレクタ、パート、タイプ、ライフライン分解を持つインスタンス名

メッセージ

メッセージとは、シグナル、メソッド呼び出し、メソッド応答の発生を意味します。メッセージには、通常 2 つのイベントがあります。1 つは送信ライフラインの送信イベント（アウト）で、もう 1 つは受信ライフラインの受信イベント（イン）です。メッセージは、水平に描画される場合と傾斜をつけて描画される場合がありますが、ダイアグラム中で、受信イベントを送信イベントより上に表示するべきではありません。

シンボル

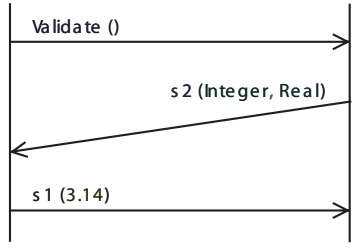

図 12: メッセージ

メッセージを送信し、受信側に渡すには時間がかかる場合がありますが、傾斜にはその意味は含まれません。同様に、水平メッセージは必ずしも直接受信側に届けられるというわけではありません。

シグナルとメッセージには関係があるため、メッセージ名はモデルに表されているシグナルを必ず参照しなければなりません。シグナルにパラメータがある場合は、メッセージには実パラメータ表現が必要です。

メッセージの作成

メッセージに関連するテキスト フィールドは 3 つあります。そのうち 1 つはシグナル名とパラメータで、あとの 2 つはゲート名です。

メッセージを配置する方法は 2 通りあります。

従来の方法：［ダイアグラム要素の作成］ツールバーの［メッセージ ライン］をクリックします。
送信者ライフラインをクリックし、次に受信者ライフラインをクリックします。

シングル クリックによる方法：［ダイアグラム要素の作成］ツールバーの［メッセージ ライン］をクリックします。
ライフラインの間で**クリックしたまま**にすると、メッセージが左側のライフラインに接続されます。ライフラインをクリックして**放す**と、右側のライフラインにメッセージが接続されます。これで、以下のような方法でメッセージを作成できます。

- クリックして放し、右のライフラインに受信ポイントを作成します。
- ドラッグしてライフラインと交差し、メッセージを受信するライフラインの左に近い位置でマウスボタンを放します。
- Shift キーを押しながらクリックして右から左にメッセージを送信します。

作成するライン タイプは以下のとおりです。

- **通常のメッセージ**：ツールバーでメッセージを選択します。ライフラインの間でクリックすると、左から右にメッセージが流れる形で作成されます。Shift キーを押しながらクリックすると、右から左にメッセージが流れる形で作成されます。クリックしたままライフラインを交差するようにドラッグし、放すと、メッセージ方向にある次のライフラインに接続します。
- **自身へのメッセージ**：要素ツールバーで、［メッセージ ライン］シンボルを選択して、同じライフラインを 2 回クリックします。
- **現在のモデル内の既存シグナルを参照して、メッセージを描画：**

1. ［ダイアグラム要素の作成］ツールバーの［メッセージ ライン］をクリックします。
2. メッセージの送信元とするライフラインにカーソルを合わせてクリックします。
3. メッセージの送信先とするライフラインの近くにカーソルを合わせて、右クリックします。ショートカット メニューの［既存要素を参照］にカーソルを合わせ、リストからシグナルを選択します。

［既存要素を参照］を選択するとスコープ内のシグナルが表示されます。以下の条件に従ってシグナルが表示されます。

- 送信先ライフラインにタイプがある場合、実現化インターフェイスのシグナルを考慮に入れて、このタイプで受信可能なすべてのシグナル、およびクラス自体で定義されたシグナルなどがリストに表示されます。
- 送信元ライフラインにタイプがある場合、すべての必須インターフェイスを考慮に入れて、このタイプで送信可能なすべてのシグナルがリストに表示されます。

- 送信元と送信先のライフラインにタイプがなく、送信先ライフラインにセレクタがある場合、実現化インターフェイスのシグナルを考慮に入れて、セレクタ タイプで受信可能なすべてのシグナルと、クラス自体で定義されたシグナルなどがリストに表示されます。
- 送信元と送信先のライフラインにタイプがなく、送信元ライフラインにセレクタがある場合、既存のすべての必須インターフェイスを考慮に入れて、セレクタ タイプで送信可能なすべてのシグナルがリストに表示されます。
- 上記以外の場合、ライフライン自体から見えるすべてのシグナルがリストに表示されます。

場合によって、メッセージが 1 本のライフラインにのみ接続されるように描画することもできます。これは、特にシーケンス図を使ってトレースする場合に有用です。メッセージには 4 つのタイプがあります。

- **新しいメッセージ**。メッセージは送信済みですが、まだ受信されていません。このメッセージは送信側に接続されています。
- **消失メッセージ**。メッセージは送信済みですが受信されていません。メッセージは送信側に接続され、小さな丸がメッセージの矢印に描画されます。
- **古いメッセージ**。メッセージは受信されましたが、送信側がまだ特定されません。このメッセージは受信側に接続されています。
- **基底メッセージ**。メッセージは受信されましたが、送信側が不明です。このメッセージは受信側に接続され、メッセージが小さな丸から出てきます。

プロパティエディタを使用してメッセージを Lost または Found とマーク付けできます。

メッセージラインは以下の方法でも作成できます :

- 1 本のライフラインを選択した状態で、Shift キーを押しながらシンボル要素ツールバーの［メッセージ ライン］シンボルをクリックすると、新しいメッセージが作成されます。新しいメッセージはライフラインの最後、消滅ライフライン シンボルの前に配置されます。
- 2 本のライフラインを選択した状態で、Shift キーを押しながらシンボル要素ツールバーの［メッセージ ライン］シンボルをクリックすると、ライフラインの間に通常のメッセージが生成されます。通常のメッセージは、ライフラインの最後（消滅ライフライン シンボルの前）に水平に、左から右の方向に配置されます。
- 1 つのメッセージを選択を選択した状態で、Shift キーを押しながらシンボル要素ツールバーの［メッセージ ライン］シンボルをクリックすると、選択したメッセージのすぐ下に通常のメッセージが生成されます。通常のメッセージは選択したメッセージと同じライフラインに接続され、同じ方向性を持ちます。

注記

メッセージを編集する際、パラメータの表示／非表示に関わらず、メッセージのすべてのパラメータを見ることができます。編集モードを終了すると、メッセージ テキストがパラメータの表示／非表示は他のメッセージと同じ設定になります。

パラメータの切り替え

ダイアグラム中のメッセージパラメータの表示 / 非表示を切り換えます。デフォルトで、すべてのパラメータが表示されます。クイックボタンを押して、現在のシーケンス図のメッセージ パラメータの表示／非表示を切り替えられます。

不完全なメッセージ

メッセージは、イベントのうち 1 つしか指定されていないという意味で、不完全な場合があります。受信イベント（イン）がない場合は、消失メッセージです。送信イベント（アウト）がない場合は、拾得メッセージです。

消失メッセージ

消失メッセージとは、送信イベントが既知で、受信イベントがないメッセージです。これを使用して、メッセージが宛先に届かないケースを記述できます。

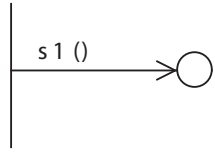

図 13: 消失メッセージ

拾得メッセージ

拾得メッセージとは、受信イベントが既知で、（既知の）送信イベントがないメッセージです。これを使用して、メッセージの送信元が記述の範囲外にあるケースをモデリングできます。また、複数のライフラインが送信側となり得、かつどのライフラインであるかがシナリオに関係しないような場合に、不要な指定を避けるためにも使用できます。

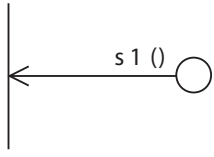

図 14: 拾得メッセージ

メッセージのコピー

メッセージをコピーするには 2 つの方法があります。1 番目の方法は常に送信者と受信者を維持します。

CTRL + ドラッグ : CTRL キーを押しながら、コピーしたいメッセージをクリックしてコピー先の場所へドラッグアンドドロップします。そして、CTRL キーをリリースします。

2 番目の方法は、送信者と受信者を変更します。

コピー / 貼り付けコマンド : コピーしたメッセージを右クリックして、メッセージのショートカットメニューを開きます。メニューから［コピー］を選択します。メッセージのコピー先の場所で右クリックして、メニューから［貼り付け］を選択します。コピー / 貼り付けのショートカット操作である、CTRL + C キーと CTRL + V キーも使用できます。貼り付けられる場所は貼り付けコマンドを発行する直前にクリックした場所です。

以下のオプションがあります。

- クリックした場所が 2 つのライフラインの間である場合、新しいメッセージはそのライフラインを結ぶように作成されます。
- クリックした場所が 2 つのライフラインの間でない場合、送信者と受信者は元のメッセージとおなじになります。

タイマー イベント

タイマーは通常、相互作用内の 2 つの異なるイベントで記述されます。1 つ目のイベントはタイマー設定で、2 つ目のイベントはタイムアウトまたはリセットです。

タイマーを使用するには、メッセージで対応するシグナルや操作を宣言する必要があるのと同様に、宣言が必要です。タイマーは、クラス図ではタイマーシンボルで宣言します。

タイマーイベントシンボルには、名前とパラメータのための 1 つのテキストフィールドがあります。

タイマー設定

設定イベントにより、タイマー インスタンスが生成され、アクティブになります。タイマー設定イベントは、タイマー設定アクションにマッピングします。

タイマー リセット

リセット イベントは、アクティブなタイマーを中止します。タイマー リセット イベントは、タイマー リセット アクションにマッピングします。

タイマー タイムアウト

タイムアウト イベントは、タイマー継続時間が経過し、タイマー シグナルが受信され、状態機械に処理されると発生するものです。タイムアウト イベントは、タイマー シグナル処理にマッピングします。

シンボル

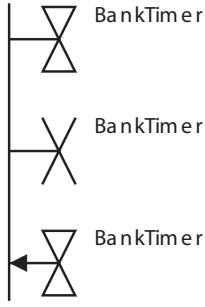

図 15: タイマー設定、リセット、タイムアウトのシンボル

参照

タイマー
タイマー設定アクション
タイマー リセット アクション

タイマー仕様ライン

タイマー仕様ラインを使用して、**絶対時間**ライン、**相対時間**ラインと**全体順序**ラインを生成します。

絶対時間ライン

絶対時間ラインをライフラインの左または右側に追加して、絶対時間、または時間範囲「{<Time>}」を指定できます。ラインはライフラインに沿って上下に移動できます。絶対時間ラインを作成するには、シンボル パレットの［タイマー仕様ライン］シンボルをクリックして、一端のみライフラインに接続したラインを描画します。

相対時間ライン

相対時間ラインを作成するには、シンボル パレットの［タイマー仕様ライン］シンボルをクリックして、両端がライフライン接続したラインを描画します。

特定の持続時間の観察「{<Duration>}」、または、持続時間の制約「{<Duration>..<Duration>}」を、テキスト フィールドで指定できます。

相対時間ラインには、上限、下限、持続時間を指定できます。ラインはライフラインの右側に描画されますが、左側に移動することもできます。相対時間ラインの上限と下限はライフラインに沿って上下に移動できます。

通常、相対時間ラインの開始イベントと停止イベントはライフラインの他のイベントに接続されます。たとえば：

- メッセージの到達
- メッセージの送信
- 参照シンボルの始点／上部
- 参照シンボルの終点／下部

相対時間ラインの始点または終点を、開始イベントと終了イベントが接続されていない他のイベントに配置できます。

全体順序ライン

全体順序ラインは 2 本のライフライン間を移動するタイマー仕様ラインです。全体順序ラインを作成するには、シンボル パレットの［タイマー仕様ライン］シンボルをクリックして、2 本のライフラインの間にラインを描画します。

全体順序ラインは、メッセージ ラインを使用しない複数の異なるライフラインでイベントを指定する際に、活用します。全体順序ラインは、中央に矢印が付いた破線で表示されます。通常、テキストはラインに関連付けられていませんが、特定の持続時間「<Duration>}」、または、時間範囲「<Duration>..<Duration>}」をラインに関連付けることができます。

ステート

ステート シンボルは、ライフラインで記述されたインスタンスが特定のステートにあることを示すのに使用します。

シンボル

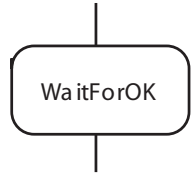

図 16: ステート

シナリオの作成時にステートを使用するのは、多くの場合、特定の状態を強調するためです。通常、ライフライン上に通過済みのステートをすべて表記することはありません。

ステートは、ライフラインが参照するアクティブ クラスの状態機械に同じ名前のステートがある場合、モデル要素に結び付けられます。

ただし、トレースの場合は、各ステート シンボルは状態機械遷移の特定の「次のステート」オカレンスにマッピングされます。これは、ライフライン オブジェクトに主状態機械が 1 つしかない場合です。パートのあるアクティブ オブジェクト、つまり複数の状態機械のあるアクティブ オブジェクトの場合は、単純なマッピングは実行不可能です。

アクション

アクション シンボルは、ライフラインで発生するイベントを表現するために使用します。状態機械のアクション シンボルに相当します。非形式的な文は、コメントとして記述する必要があります。

シンボル

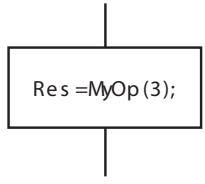

図 17: アクション

アクションで使用できるテキスト構文は、状態機械図のアクション（タスク）シンボルと同じです。

生成

生成イベントは、アクティブ クラスに適用される New 操作に相当します。

生成されたライフラインは、生成イベント受信前は破線で示され、まだ生成されていないことを示します。生成ライン上の名前はライフラインに対応するクラスの名前です。

シンボル

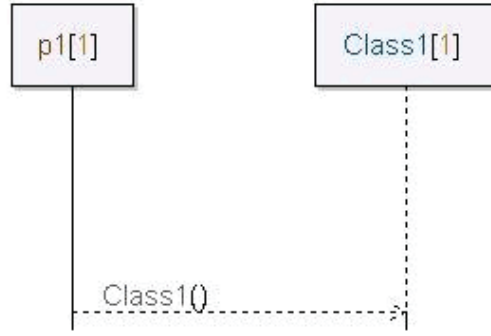

図 18: 生成メッセージ

作成ラインの作成

クラスの動的インスタンスを表現するライフラインを描画する際、作成イベントを描画できます。これは［ダイアグラム要素の作成］ツールバーの［生成ライン］ボタンを使用して行い、メッセージのように扱うことができます。生成ラインの名前は、ライフラインに対応するクラスの名前にします。生成ラインは、クラスのコンストラクタ操作を参照します。生成ラインに関連するテキスト フィールドが 3 つあります。そのうち 1 つはコンストラクタ操作名とパラメータで、あとの 2 つはゲート名です。仮パラメータは、操作パラメータと同様に、メソッド呼び出しラインに追加できます。

コンストラクタのバインディング

基底クラスのコンストラクタが「initialize」という名前になっていると、基底クラスのコンストラクタへのコンストラクタ初期化参照のバインディングが失敗します。コンストラクタ名はクラス名と同じにすることを推奨します。

例 9: バインドされないコンストラクタ initialize

```
class AutoDispatchableClass :tor::DispatchableClass {
  initialize(tor::DispatchableClass d) {
    d.addToCurrentDispatcher(this);
    init();
    'start'();
  }
}

class MyClass :AutoDispatchableClass {
  initialize(tor::DispatchableClass
d):AutoDispatchableClass(d) {  }
}
```

AutoDispatchableClass 参照はバインドされない。

消滅

消滅イベントは、インスタンスの終了を表します。状態機械の停止アクションに相当します。消滅イベント後は、ライフラインにイベントは発生しません。

シンボル

図 19: 消滅

インライン フレーム

インライン フレーム シンボルを使用すると、相互作用内で同様に処理すべきメッセージをグループ化できます。つまり、バリエーションごとにダイアグラムを作成するのでなく、複数の種類のバリエーションを 1 つのダイアグラムで表現できます。

シンボル

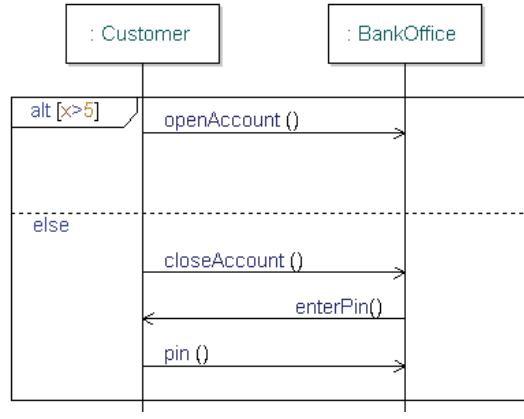

図 20: インライン フレーム

[インライン フレーム] シンボルを別の [インライン フレーム] シンボル内に保持することが可能です。既存の [インライン フレーム] シンボルと同じ高さに 2 つ目の新しい [インライン フレーム] シンボルを追加すると、既存の [インライン フレーム] シンボルの内部に入ります。

[インライン フレーム] シンボルには、1 つまたは複数のインライン フレーム セクションがあります。デフォルトの [インライン フレーム] シンボルにはインライン フレーム セクションが 1 つあります。インライン フレーム セパレータ ラインは、[インライン フレーム] シンボルを複数のインライン フレーム セクションに分割します。個々のインライン フレーム セパレータ ラインには制約テキストがあります。

インライン フレーム セパレータ ラインは、[インライン フレーム] シンボルが 1 つだけ選択されたときに表示されるライン ハンドルとともに、作成されます。

インライン フレーム セパレータ ラインはシンボル内で上下にドラッグできますが、同じインライン フレーム シンボルに接続された、他のセパレータ ラインを越えることはできません。インライン フレーム セパレータ ラインを削除するには、セパレータ ラインを選択して、Delete キーを押します。

セクションを削除すると、セクション内のオブジェクトも削除されたセパレータの一部として削除されます。

インライン フレームには、以下の組み合わせのテキストが 1 つあります。

- 演算子キーワード 例 : seq （デフォルト キーワード）alt、else、loop、assert
- 制約テキスト 例 : ”[a<3]”、”else”

バリエーション

バリエーションはいくつか考えられます。フレームはメッセージの代替グループを表現するために分割されることもあります。バリエーションを以下に示します。

- `alt`：代替の 1 つのブランチ、つまり分岐を表します。フレームは複数のオペランドに分割でき、各オペランドを条件と関連付けることができます。条件の値が true の代替ブランチのみが選択されます。ブランチのうち 1 つのみを else ブランチにできます。
- `opt`：グループ化されたメッセージがオプションであることを表します。つまり、発生しなくてもかまいません。opt フレームは分割できません。条件と関連付けることはできます。この場合、2 つ目の選択肢が空であるような選択肢と同じような振る舞いをします。
- `loop`：一連のメッセージが、何度か繰り返されることを表します。loop フレームは分割できません。繰り返し回数は、最小値と最大値を `loop(min, max)` という形式で指定します。「`max`」に「*」指定することもできます。これは、無限ループを表します。
- `par`：複数のオペランドのメッセージを互いにインターリーブするか、並行して発生するようにすることはできるが、各オペランド内の整列制約を守る必要があることを表します。有効にするには、par フレームを分割する必要があります。
- `seq`：これは、シーケンス図の通常のセマンティックを表します。つまり、各ライフラインは他のライフラインと独立です。まず弱い順序付けが厳密な順序付けに優先して使用されます。
- `strict`：シーケンス図または組み合わせフラグメント内の該当メッセージに、厳密な順序付けをする必要があることを表します。つまり、ダイアグラム中の垂直方向の位置は、事象が発生する順序に相当します。これを、シーケンス図のデフォルトである、各ライフラインにそれぞれのタイムラインがある弱い順序付けと比較します。厳密な順序付けを使用すると、これを関連ライフラインの共通グローバル時間と考えることができます。
- `neg`：該当の一連のメッセージが無効であることを表します。
- `critical`：該当のメッセージが他のインライン フレームとインターリーブできないことを示します。これはたとえば、par フレーム内で、一連のメッセージの暗黙的インターリーブを無効にするために使用できます。
- `break`：シーケンス図の以降の部分に割り込み、代わりに break フレームで囲まれた一連のメッセージを実行するという例外的なオカレンスを表します。break フレームは分割できません。
- `assert`：assert フレームで表されたシーケンスのみが有効で、他のシーケンスは無効であることを表します。assert フレームは分割できません。
- `ignore`：所定のメッセージが重要でなく、フレームに表示されないことを表します。これで、相互作用で最も重要なメッセージのみが表示されるようになります。形式は、ignore {<list_of_messages>} です。逆の操作は consider です。ignore フレームは分割できません。
- `consider`：フレーム内で所定のメッセージが重要であり、表示されないメッセージが重要でないことを表します。形式は、consider {<list_of_messages>} です。逆の操作は ignore です。consider フレームは分割できません。

共通リージョン

共通リージョンは、1 つのライフラインでメッセージが（送信または）受信された順序が重要でないことを示すのに使用します。

シンボル

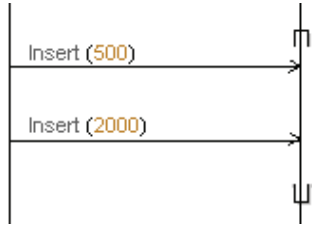

図 21: 共通リージョン

継続

継続は alt インライン フレームでのみ使用され、シーケンスのパートからパートへの継続のしかたを決定するラベルとして機能します。継続で終わる相互作用は、同じ継続で始まる相互作用のみ継続されます。

シンボル

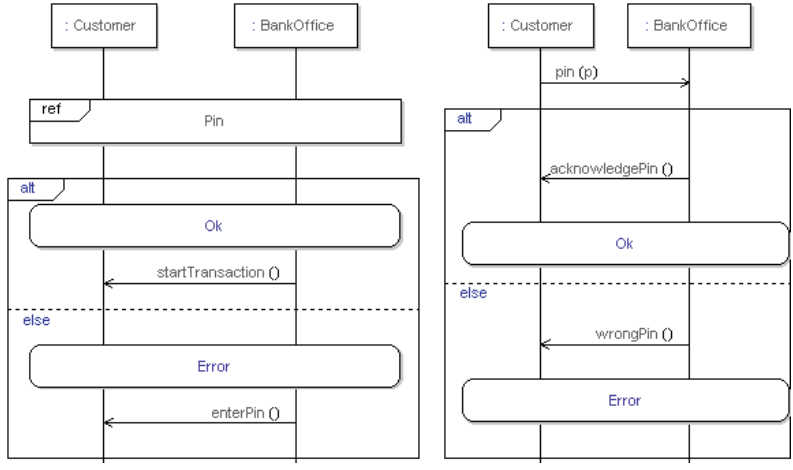

図 22: 継続

継続シンボルは、ステート シンボルと似ていますが、複数のライフラインにまたがることができます。

このシンボルの中央にはテキスト フィールドがあります。入力されたテキストは解析されず、そのままシンボルに保存されます。

［継続］シンボル内にシンボルとラインを配置することはできません。

メソッド呼び出し

メソッド呼び出しはメッセージに似ていますが、常に同期的です。つまり、必ずメソッド応答と関連付けられます。メソッド呼び出しは、たとえば、異なるクラス間での操作の起動方法などのモデリングに使用します。

シンボル

図 23: メソッド呼び出しとメソッド応答

メソッド呼び出しの結果は、実線の矢印で表される呼び出し、破線の矢印で表される応答、アクティベーション領域、サスペンション領域の 4 つの図形要素で示されます。サスペンション領域は呼び出すライフラインの破線の長方形で、アクティベーション領域は呼び出されるライフラインの実線の長方形です。

呼び出しメッセージと応答メッセージには、関連するテキスト フィールドがそれぞれに 3 つあります。そのうちの 1 つは操作名とパラメータで、あとの 2 つはゲート名です。

完全なメソッド呼び出しを描画するには以下の手順を行います。

1. ［ダイアグラム要素の作成］ツールバーの［メソッド呼び出し］シンボルをクリックします。
2. メソッド呼び出しの送信側のライフラインに**呼び出しメッセージ**開始イベントを置き、これを受信側にドラッグします。
3. 操作名とパラメータ情報を入力するか、操作をモデルからメッセージにドラッグします。
4. 呼び出しメッセージと応答メッセージの名前フィールドの操作パラメータ タイプ情報を編集します。

応答メッセージのメイン テキストは、通常、呼び出しメッセージと同じメソッドを参照します。メソッドが送信パラメータに値を割り当て、さらに戻り値があるような場合、パラメータは異なります。応答ラインのみに戻り値 < 値 > を与えることはできます。

メソッド呼び出しまたは応答ラインを削除すると、これに接続されたサスペンションとアクティベーション領域シンボルも削除されます。

サスペンションまたはアクティベーション領域シンボルを削除すると、シンボルのみ削除され、接続されたメソッド呼び出しと応答ラインは削除されません。

呼び出しメッセージをドラッグすると、メソッド呼び出しシンボル全体が対応する方向に移動します。

応答メッセージをとラッグすると、一致する方向に、メソッド呼び出し シンボルのサイズが縮小または拡大されます。

注記

メソッド呼び出しを切り取り／貼り付け、またはコピー／貼り付けによって編集すると、予期せぬ結果になることがあります。貼り付け操作時のエディタの振る舞いには、アクティベーションとサスペンション領域シンボルは含まれません。領域シンボルはオプションと考えることができます。領域シンボルを選択して Delete キーを押すと、メソッド呼び出し全体を削除せずに、これらのシンボルを削除できます。領域シンボルを元に戻すには、取り消しを実行するか、新しいメソッド呼び出しを生成します。

ゲート名

ショートカット メニューで［ゲート テキストの追加／削除］を選択して、メッセージ、メソッド呼び出し、または作成イベントにゲート名を追加できます。2 つのゲート名テキストはラインの下に配置されます。ゲート テキストがアクティブになると、ゲートがデフォルト名を取得します。デフォルト名は変更可能です。

アクティベーションとサスペンション

メソッド呼び出しを発信するライフラインは、受信側が実行でビジー状態の間は中断されます。つまり、応答を待つのみです。メソッド呼び出しを受信するライフラインは、起動されたメソッドの実行中はアクティブになります。呼び出し側に応答が返されると、アクティベーション領域とサスペンション領域の両方がクローズになります。

表示と削除フィルタ

レイアウトの圧縮

［レイアウトの圧縮］ボタンは、［オプション］のシーケンス図の指定にしたがって、メッセージ間とライフライン間の間隔を圧縮します。

［レイアウトの圧縮］ボタンをクリックすると、ライフラインが圧縮されて、水平方向に移動します。

Shift キーを押しながら［レイアウトの圧縮］ボタンをクリックすると、ライフラインが上記のように圧縮され、さらにライフラインのオブジェクトもライフラインに沿って上／下に圧縮されます。

Ctrl キーを押しながら［レイアウトの圧縮］ボタンをクリックすると、ライフラインは最初のイベントを持つライフラインがダイアグラムの左になるように再配置されます。

Shift + Ctrl キーを押しながら、［レイアウトの圧縮］ボタンをクリックすると、ライフラインのライフラインとオブジェクトが上記のように圧縮され、さらにライフラインと最初のイベント（たとえばシグナル送信）が関連付けられ、ダイアグラムの左側に再配置されます。

選択したシグナルの削除

選択したメッセージを削除します。このコマンドは、選択されたメッセージと同じシグナルを使用して、メッセージを削除します。また、他のオブジェクトの削除にも使用できます。

<X> が選択された場合、このコマンドですべての <X> を削除します。

<X> には以下のメッセージが適用されます。

- 生成ライン
- ステート シンボル
- タイマー シンボル（設定、リセット、タイムアウト）
- タイマー仕様ライン（絶対時間、相対時間、全体順序ライン）
- メソッド呼び出し（呼び出しライン、アクティベーション シンボル、応答ライン、サスペンション シンボル）
- アクション シンボル
- 消滅シンボル
- 参照シンボル
- インライン フレーム シンボル
- 継続シンボル
- テキスト シンボル
- コメント シンボル

選択したシグナルの保持

Shift キーを押しながら［選択したシグナルの削除］をクリックすると、フィルタの効果が逆になります。選択したメッセージと、選択したメッセージと同じシグナルを使用するメッセージだけがシーケンス図に保持されます。他のオブジェクトについては、以下のルールに従います。

<X> を選択していない場合（<X> は選択したシグナルの削除で定義される）、このコマンドによりすべての <X> が削除されます。

スペースの確保

このコマンドを実行すると、選択したシンボルまたは行の下にスペースが作成されます。Shift キーを押しながらツールバーの［スペースの確保］をクリックすると、選択したシンボルまたは行の下のスペースが削除されます。

相互作用概観図

相互作用概観図は、相互作用間の制御フローに焦点を当てたアクティビティ図の一形態です。

相互作用概観図内の相互作用参照は、操作とアクティビティを定義、参照できます。相互作用参照は、アクション ノード ノードとオブジェクト ノードの代わりに使用されます。アクティビティ エッジと分岐ノード、フォーク ノード、アクティビティ終了ノードなどの構成要素は、アクティビティズと同じです。

次の表に、バリエーションに示された一般的な相互作用オペランドを相互作用概観図で表す方法を示します。

オペランド	相互作用概観図の構成要素
alt	対応するマージ ノードに一致する分岐ノード
par	対応するジョイン ノードに一致するフォーク ノード
loop	ダイアグラム中の分岐とグラフ サイクル

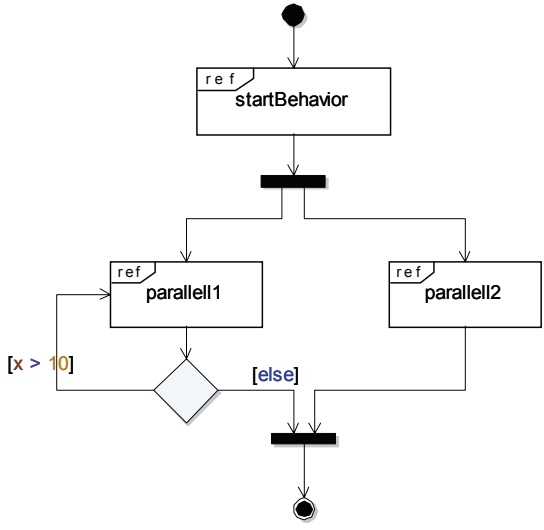

図 24: 相互作用概観図

相互作用概観図の作成

相互作用概観図はクラスとユース ケースに含めることができます。

1. ［モデル ビュー］のクラス（ユース ケース）を選択します。
2. ショートカット メニューから［新規］を選択し、［相互作用概観図］を選択します。

相互作用概観図のモデル要素

相互作用概観図には、以下の要素が使用されます。

- 分岐
- フロー終了
- フォーク
- 開始ノード
- ジョイン
- マージ
- 相互作用参照、アクションノードを参照。
- 関係

参照

シーケンス図

アクティビティ図

パッケージ モデリング

比較的大きなシステムのモデリングの場合は、すべての定義を論理的で管理できるグループとして体系付けるには、パッケージ構成要素が不可欠です。体系付けの原則として、同時に変更される可能性がある、意味的に近接した要素をグループ化するとよいでしょう。

パッケージ図

パッケージ図は、パッケージの集合とその相互の関係を可視化するために使用します。システムの内訳を論理パッケージとパッケージ間の依存関係にモデリングするために使用します。

パッケージ図には、パッケージと、インポート依存やアクセス依存のようなパッケージ間の依存が含まれます。

同じ目的でクラス図も使用できます。

例

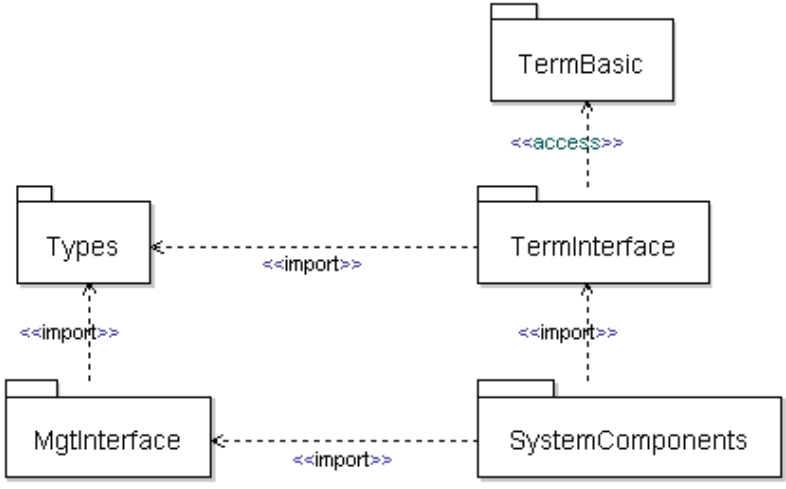

図 25: パッケージとその関係

パッケージ図のモデル要素

パッケージ図には、以下の要素が使用されます。

- パッケージ
- 関係

参照

クラス図

パッケージ

パッケージは、要素をグループに体系付ける仕組みです。パッケージは、グループ化された要素の名前空間となります。パッケージ内では、要素はそれぞれの名前を使用して直接参照できますが、パッケージ外からは、多くの場合、モデル要素の名前を修飾して参照する必要があります。

通常、モデルは互いに依存する複数のパッケージから構成されます。パッケージ相互の関係を理解することは、システムが複雑か複雑でないかにかかわらず、モデリングという点で重要です。パッケージの相互関係は多くの場合システム アーキテクチャを反映したものとなるため、システムの規模が大きくなるにつれ、この重要性は高くなります。

シンボル

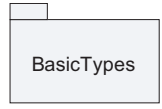

図 26: パッケージ

パッケージにより、パッケージで定義された個々の要素の可視性、要素へのアクセス権限も制御できます。

- クラスやその他のパッケージなどの定義は、パッケージに収集できます。
- パッケージはインポートされることも、別のパッケージからアクセスすることもできます。

他のシンボル階層をパッケージ シンボル内にネストできます。パッケージシンボル内に作成された要素の所有者は、そのパッケージになります

構文

パッケージ シンボルには、パッケージの名前の入るテキスト フィールドが含まれます。参照対象のパッケージが別の名前空間で定義されている場合は、パッケージ名の前には OuterPackage::MyPackage などのように修飾子が付きます。

参照

関係

関係

パッケージ図では以下の関係を使うことができます。詳細については UML の関係で説明しています。

- 依存
- 包含

依存はステレオタイプ化されて、より正確な意味を与えられます。このために使用するステレオタイプは、<<import>> と <<access>> です。

インポート

インポートは、特にパッケージ間や、クラスまたは状態機械などからパッケージへの場合で有効な特殊依存です。定義の名前をパッケージから現在の名前空間にインポートします。通常は、現在の名前空間もパッケージです。これで、修飾子を使用する必

要がなくなります。パッケージ Q によってインポートされたパッケージ P 内の定義の名前は、次にパッケージ Q をインポートするかパッケージ Q にアクセスするパッケージに自動的に含められます。

図 27: インポート

注記

インポート スコープ内で修飾子なしにアクセスできる名前の数が非常に大きくなるので、インポート依存関係の使用は制限してください。アクセス依存関係を使用した方がよいでしょう。定義の一部だけを使用する場合は、修飾子の使用も考慮する必要があります。修飾名を使用すると入力する文字数が多くなりますが、モデルに使用されている定義がわかりやすくなります。

アクセス

アクセスは、特にパッケージ間や、クラスまたは状態機械などからパッケージへの場合で有効な特殊依存です。定義の名前をパッケージから現在の名前空間にインポートします。通常は、現在の名前空間もパッケージです。これで、修飾子を使用する必要がなくなります。パッケージ Q がアクセスしたパッケージ P 内の定義の名前は、次にパッケージ Q をインポートするかパッケージ Q にアクセスするパッケージには含められません。

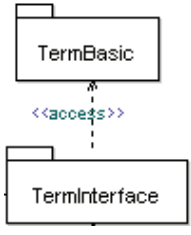

図 28: アクセス

インポートとアクセスは、密接な関係にあります。主な違いは、インポートは遷移的である点です。つまり、パッケージにアクセスしたりパッケージをインポートしたりすると、そのパッケージが次にインポートする定義の名前も自動的に取得されますが、アクセス先の定義の名前は取得されません。166 ページの図 25 では、パッケージ **TermBasic** はパッケージ **TermInterface** のアクセス先となっています。つまり、**TermBasic** の定義の名前を **TermInterface** で直接参照できます。ただし、これらの名前は、パッケージ **TermInterface** をインポートするパッケージ **SystemComponents** で

は直接には使用できません。したがって、**SystemComponents** では、明示的にパッケージ **TermBasic** をインポートまたはアクセスして名前を参照するか、明示的に名前を修飾する必要があります。

アーキテクチャの観点では、インポートよりもアクセスのほうが好ましいと言えます。これは、必要なすべてのパッケージを検討することがアクセスによって強制され、誤って余分なものの取り込むことがなくなるからです。

注記

パッケージをインポートしたりパッケージにアクセスしたりしなければ、パッケージ内の定義を参照できないわけではありません。定義が public であれば、修飾を使用して参照できます。たとえば、`TermBasic::Xterm` を使用してパッケージ `TermBasic` の要素 `Xterm` を参照できます。しかし、明快さという点で、通常はパッケージの相互依存関係の記述を作成するほうがよいでしょう。

参照

UML の関係

<<noScope>> パッケージ

<<noScope>> パッケージは、通常、パッケージの要素を複数のファイルに分割する必要がある場合に使用します。ただし、パッケージの内容を複数のパートに構造化する必要があるが、UML の名前スコープ ポイントのパッケージは 1 つのエンティティとして見なければならない場合にも使用できます。

セマンティック上は、<<noScope>> ステレオタイプのパッケージは、[モデル ビュー] の他のパッケージと同じ可視性を持ち、別のファイルに格納するという点でも他のパッケージと同じように機能します。セマンティックの観点では、<<noScope>> パッケージのすべての要素は包含するパッケージの一部と見なされます。修飾子を使用して <<noScope>> パッケージ内の要素を参照する場合、<<noScope>> パッケージの名前は修飾子の一部として使用できません。<<noScope>> ステレオタイプは、修飾子なしで、すべての定義をパッケージ外部でも見えるようにします。また、明示的な修飾子を使用して曖昧なケースを解決することもできます。

例 10: <<noScope>> **パッケージ**

```
package A {
    <<noScope>> package B {
        class C {

        }
    }
    C c; // <<noScope>> makes C visible
}

package A {
    <<noScope>> package B {
        class C {

        }
    }
    class C { }
```

```
    C c; // class A::C hides class B::C
}

package A {
    <<noScope>> package B1 {
        class C {

        }
    }
    <<noScope>> package B2 {
        class C {

        }
    }
    B1::C c;
/* 'C' is an ambiguous name. B1::C or B2::C must be used. If
C is used without qualifier there will be name resolution
errors. */
}
```

<<openNamespace>> パッケージ

複数のパッケージをまとめて 1 つのパッケージとして定義し、なおかつ含まれるパッケージの数を段階的に増やすことができると便利な場合があります。この場合、あるセッションでどのサブパッケージをロードしているかによって、グループ化しているパッケージの内容は変化します。

これを実現するには、DOORS Analyst では <<openNamespace>> パッケージを使用します。動作シナリオは以下のとおりです。同じスコープ内で 2 つのパッケージを、たとえばモデル ルートとして、定義します。パッケージに同じ名前を付け、両方とも <<openNamespace>> ステレオタイプにします。セマンティック上は、これでパッケージの内容はマージされます。この指定で、一方のパッケージの要素からもう一方のパッケージの要素を修飾子なしで直接使用でき、名前はマージされるすべてのパッケージ内で一意となる必要が生じます。

ネストされた <<openNamespace>> パッケージの階層を持つことができます。したがって、たとえば 1 つのファイルに格納された、<<openNamespace>> Sub を含む <<openNamespace>> Top がある場合、<<openNamespace>> Sub を持つ <<openNamespace>> Top を含む別のファイルを持つことができます。これらのファイルを両方とも同じプロジェクトにロードすると、Top の内容と Sub の内容がマージされます。

<<openNamespace>> パッケージを使用する最も重要なシナリオは、別に管理されているパッケージ階層のベース バージョンがあり、それを特定のアプリケーションで使用する際にサブパッケージなどで拡張したい場合です。

クラス モデリング

クラス モデリングは、設計中のシステムを構成するオブジェクトの種類を特定するプロセスです。この作業は、多くの場合、設計フェーズの初期や分析フェーズで行いますが、通常は、ユース ケースおよび／またはシナリオ モデリングを通じて、設計対象

システムを構成するオブジェクトが特定された後に行います。他のオブジェクトと同じプロパティ、振る舞い、関係を持つと考えられるオブジェクトは、1 つのグループにまとめられ、オブジェクトの「クラス」としてモデリングされます。

クラス モデリング作業には、クラスを同定する作業の他に、これらのクラスを定義する作業も含まれます。この作業には、クラス図を使用します。同定されたクラスごとに、以下の質問に対する回答を検討します。

クラスに構造があるか。
クラスのインスタンスにはどのパートが含まれているか。

クラスの構造は、属性や、汎化や関連などの関係を使用してクラス図に記述されます。クラスがどのように構成されているかを示すためには、合成構造図も使用できます。

クラスに振る舞いがあるか。
どの操作が可能か。

クラスの振る舞いはクラスに対する操作と見なされ、これらの操作のシグニチャをクラス図に記述します。シグナル、タイマー、状態機械など、クラスの他の振る舞い特性に関しても同じことが当てはまります。

クラスと他の要素の間にどのような関係があるか。

クラスには、他のクラスとの関係に加え、インターフェイス、データ型、選択などとの関係もあります。クラス モデリングでの使用方法の詳細は、265 ページの「UML の関係」セクションを参照してください。

クラスはアクティブかパッシブか。

単純に言うと、アクティブ クラスは動的なイベント起動の振る舞いを定義し、パッシブ クラスは情報を処理します。アクティブクラスのインスタンスは、イベントをディスパッチできます。

クラスが環境に露出している通信ポートはどれか。

クラスのポートは、クラス図で可視化されます。

クラス図

クラス図はモデルを静的に表示するもので、モデル内のオブジェクトのタイプを記述するために使用します。これらのタイプは通常はクラスですが、基本、列挙、インターフェイス、選択、シンタイプなど他の分類子の場合もあります。クラス図により、タイプ間の関係と、その構造特性や振る舞い特性も示されます。

クラス図に表示される定義は、デフォルトでは、ダイアグラムを持つスコープ（クラスやパッケージなど）に含まれますが、別のスコープの定義を表示することもできます。

パッケージ図を、システムのパッケージやその相互依存関係の記述手段として使用する方法については 165 ページの「パッケージ モデリング」を参照してください。ただし、同じ情報をクラス図で記述することもできます。

クラス図の例

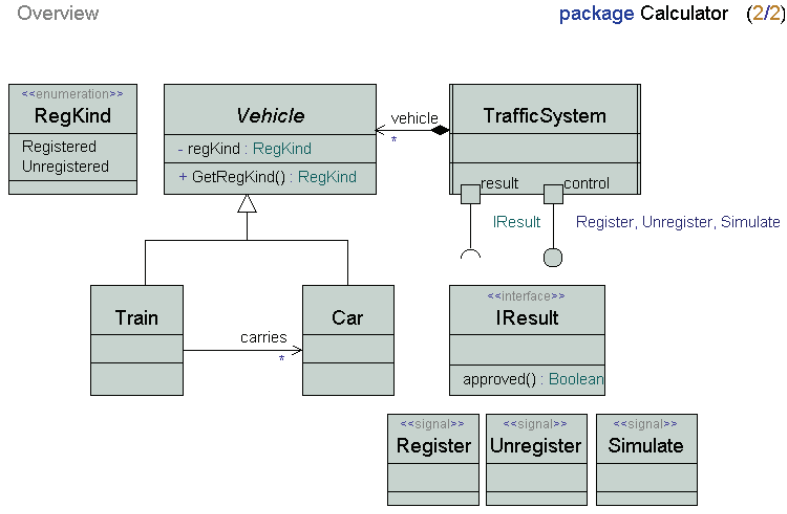

図 29: クラス図

クラス図のモデル要素

クラス図には、以下のモデル要素が使用されます。

- アーティファクト
- コラボレーション
- クラス
  - アクティブ クラス
- 属性
- 操作
- ポート
- インターフェイス
  - 実現化インターフェイス
  - 要求インターフェイス
- シグナル
- シグナル リスト

- タイマー
- データ型
- 選択
- シンタイプ
- 状態機械
- 関係

参照

パッケージ図

クラス

クラスは、同じプロパティ（属性）、振る舞い（操作）、構造、関係を持つオブジェクトのグループの抽象化です。クラスは、（抽象と定義されていなければ）複数のインスタンスにインスタンス化できます。それらはすべて同じプロパティを持ちます。

シンボル

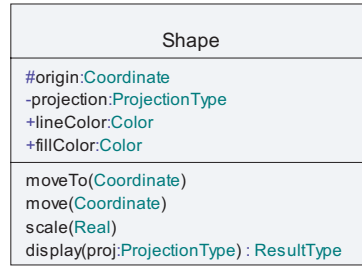

図 30: 属性と操作を持つクラス

クラスのインスタンスがそれ自体の実行スレッドを維持する（他のインスタンスと並行して実行される）場合、クラスはアクティブ クラスであると言います。そうでなければ、インスタンスは別のアクティブ インスタンスのスレッドで実行され、この場合、クラスはパッシブであると言います。

クラスをアクティブにするには、以下のいずれかを行います。

- ダイアグラム（または［モデル ビュー］）で、アクティブにしたいクラスを右クリックして、ショートカット メニューの［アクティブ］をクリックします。
- クラスのプロパティ エディタを表示して、［Active］を選択します。

アクティブ クラスは、ダイアグラム内で、外枠の縦が二重線で表示されます。

クラスには、内部通信から見た構造を記述する合成構造図（旧アーキテクチャ図）にパートとコネクタで可視化された内部構造を持たせることもできます。また、ランタイム実行の観点で記述する状態機械（initialize またはクラスと同じ名前で呼ばれる）を使用することもできます。この状態機械は、アクティブ クラスのインスタンスの生成時に実行される予定の「メイン」の振る舞いです。

また、クラスには一連のポートも使用できます。ポートは、クラスのアーキテクチャ記述で、クラスのインスタンスがどのように他のインスタンスと接続できるかを指定します。ポートは、複数の関係者に露出されている一連のインターフェイスのグループ化にも使用されます。

モデルにクラスを追加するには、複数の方法があります。

- ワークスペース ウィンドウの［モデル ビュー］にクラスを直接追加します。クラスが常駐するスコープを選択して、ショートカット メニューから［新規］、［クラス図］を選択します。
- クラス図にクラスを描画します。クラス図を作成して開き、ツールバーから［クラス］シンボルを選択してダイアグラム内に配置します。
- 合成構造図で、タイプ名のないバインドされていないパートをダブルクリックします。この操作で、該当するパートを表示するダイアグラムを新たに作成できます。このダイアグラムは、パートに対して作成されたインライン クラスに属します。作成可能なダイアグラムは、クラス図、合成構造図、状態機械図、ユースケース図です。

アクティブ クラスの複数状態機械

アクティブ クラスに任意数の状態機械を挿入できます。ただし、以下の点を考慮に入れる必要があります。

- 状態機械の 1 つに **initialize** の名前が付いている場合、またはクラスと同じ名前が付いている場合、この状態機械はクラスのメイン 状態機械と見なされます。クラスのインスタンスの作成時、この状態機械が実行されます。この状態機械を省略すると、ビルド時にスタートとストップ シンボルが状態機械図に自動挿入されます。
- アクティブ クラスの他の状態機械を実行する場合は、明示的に呼び出す必要があります。

構文

クラスシンボルには、以下に示すように、編集可能なテキスト フィールドのある入力領域があります。

- クラス ヘッダー（必須）
- 属性（オプション）
- 操作（オプション）
- 制約区画（オプション）
- ステレオタイプ インスタンス区画（オプション）

クラス ヘッダー

以下の例に、いくつかのタイプのクラス ヘッダーを示します。

例 11: クラス ヘッダー

単純なクラス：

```
myClass
```

仮想性を含むクラス：

```
redefined myC
```

テンプレート パラメータを使用するクラス：

```
MyParamClass < type T, Integer c >
```

属性

例 12: クラスと属性

属性を持つクラス：

```
public A :Integer = 4
```

多重度を持つ属性：

```
A:Integer [10]
B:Integer [3,>15]
C:Integer [*]
```

操作

例 13: シグナル

```
signal s (Integer, Real)
```

例 14: メソッドの例

```
private m( x:Integer) :Integer
```

抽象クラス

クラスは「抽象」である場合があります。抽象クラスのインスタンスは生成できません。したがって、このクラスをインスタンス化するには特化する必要があります。

クラスが抽象であれば、クラスの名前は、クラス シンボルで*イタリック*表記されます。

クラスを抽象にするには、以下のいずれかを行います。

- ダイアグラム（または［モデル ビュー］）で、抽象にしたいクラスを右クリックして、ショートカット メニューの［抽象］をクリックします。
- クラスのプロパティ エディタを表示して、［Abstract］を選択します。

仮想性

仮想性は、クラスが再定義可能かどうかを決めるものです。これは、クラスが別のクラスに包含されている場合のみ有効です。

可視性

属性や操作といったクラスの特性の可視性は、定義されているクラスの外からアクセスできるかどうかを定義するものです。

- **なし**

  特性に定義されている可視性はありません。
- **Public**

  特性は、包含されているクラスが見える場所であればとこからでも参照できます。
- **Protected**

  特性は、特性を定義するクラスのどの子孫（特化による）からでも参照できます。
- **Private**

  private 特性を定義するクラスのみが使用できる特性です。
- **Package**

  特性は、包含されているクラスが見える、包含する最も近いパッケージ内の場所であればとこからでも参照できます。

可視性の詳細については、可視性を参照してください。

外部クラス

クラスを外部として定義するには、以下の手順を行います。

- クラスのプロパティ エディタを表示して、［External］を選択します。外部プロパティは、プロパティ エディタにのみ表示されます。

クラスとコンポーネント

コンポーネントとして抽象を表現する特定の概念はありませんが、抽象は他の方法でモデリングできます。

UML では、クラスとコンポーネントはよく似ています。コンポーネントは、メタモデルのクラスのサブクラスです。クラスにもコンポーネントにも、属性、操作、合成構造（合成構造図に表されるもの）、ポート、インターフェイスなどがあります。コン

ポーネントの主な目的は、あるエンティティを表現する用語を提供すること、そして、コンポーネント ベースのモデリングで最も重要な機能性を強調することです。これには、コンポーネントの実現化を表す能力や、コンポーネントの要求インターフェイスや提供インターフェイスを指定する能力も含まれます。通常、提供インターフェイスは実現化分類子のいずれかによって実現化されます。

制約区画

［制約区画の追加］ショートカット メニュー項目を使用して、1 つまたは複数の制約区画をクラス シンボルに追加できます。制約区画はインターフェイスシンボルやステレオタイプシンボルなど、他のクラス的な シンボルにも追加できます。

制約区画は、クラス シンボルの最後の通常の表示可能入力領域の下に置かれます。

制約区画は、制約シンボルに似ています。1 つの読み取り専用 “{}” テキスト ラベルと、編集可能なメイン テキスト ラベルがあります。

ショートカットコマンド［制約を区画として表示］を使用して、クラス シンボルの下に制約区画がないクラス シンボルに関連付けられたモデル要素に適用されている制約ごとに、1 つの制約区画を作成して追加します。ショートカットコマンド［制約をシンボルとして表示］を使用して、クラスシンボルに関連付けられたモデル要素に適用されている制約ごとに、1 つの制約シンボルを作成して追加します。

ステレオタイプ インスタンス区画

［ステレオタイプインスタンス区画の追加］ショートカット メニュー項目を使用して、1 つまたは複数のステレオタイプ インスタンス区画をクラス シンボルに追加できます。ステレオタイプ インスタンス区画は、インターフェイスシンボルやステレオタイプシンボルなど、他のクラス的な シンボルにも追加できます。

ステレオタイプ インスタンス区画は、クラス シンボルの最後の通常の表示可能区画の下に置かれます。

ステレオタイプ インスタンス区画は、ステレオタイプ インスタンス シンボルに似ています。1 つの読み取り専用 ”<<>>” テキスト ラベルと、編集可能なメイン テキスト ラベルがあります。

クラス シンボルのショートカット コマンド［ステレオタイプを区画として表示］を使用して、クラス シンボルの下にステレオタイプ インスタンス区画がないクラス シンボルに関連付けられたモデル要素に適用されているステレオタイプ インスタンスごとに、1 つのステレオタイプ インスタンス区画を作成して追加します。ショートカットコマンド［ステレオタイプをシンボルとして表示］を使用して、クラスシンボルに関連付けられたモデル要素に適用されているステレオタイプ インスタンスごとに、1 つのステレオタイプ インスタンス シンボルを作成して追加します。

参照

データ型

選択

コラボレーション

コラボレーション シンボルは、アイコン モードのサポートも含め、クラス シンボルと同じように振る舞いますが、コラボレーション シンボルには属性と操作が表示されません。

属性

属性は、実行時に 1 つまたは複数の値を持つ構造特性です。

属性は、UML 言語の、関連する複数の構成要素をモデリングするために使用します。

- **属性**

  構造化分類子の属性は、属性としてモデリングされます。このような属性のインスタンスは、フィールドと呼ばれ、フィールド式を使用して参照できます。また、クラス スコープ属性 (いわゆる「静的属性」) というものもあります。あるクラスのすべてのインスタンスは、この属性について同じ値を共有します。

  合成構造図では、合成属性はパートととして参照されることがあります。これは、合成構造図がクラスの階層構造を示すものであるためです。

  属性は、関連の端を表すのにも使用します。

- **ローカル変数**

  状態機械、操作、合成文のローカル変数は、属性としてモデリングされます。このような属性は、名前で直接、必要に応じてスコープ分類子で修飾して参照できます。

- **定数**

  定数は、読み取り専用属性としてモデリングされます。定数の値は、属性のデフォルト値です。通常、定数はパッケージ レベルで定義されますが、属性の定義できる場所であればどこでも定義できます。定数は、名前で直接参照されますが、必要に応じてスコープ分類子で修飾して参照することもできます。定数の値は、いったん設定すると変更できません。

1 つの属性は、必ず 1 つの静的なタイプをもちます。タイプは、属性の定義の時点で決定され、以下のいずれかになります :

- クラス
- インターフェイス
- 基本型または列挙型
- シンタイプ
- デリゲート
- 選択

属性は、関連と密接な関係があります。誘導可能な関連の端と属性は、実質的には同じものです。つまり、まず属性を定義し、次にこの属性を、誘導可能な関連の端の役割名としてクラス図に可視化できる、ということです。逆ももちろん可能です。まず、誘導可能な関連の端が 1 つある関連を定義します。次に、関連の端を、クラス シンボルの属性入力領域に属性として可視化します。

呼び出しをするために、特定の関連の端や関連付けられている属性を使用する場合は、誘導可能でなければなりません。

例 15: 誘導可能性

A と B というクラスがあるとします。操作 B.op() をクラス A から起動しようとしているものとします。

関連の端の名前（役割名）が b で、向きが A から B の関連がある場合、関連が誘導可能である場合のみ、b.op(); を呼び出すことができます。

属性は、合成構造図でシンボルとして可視化できます。これは、包含クラスのすべての属性について可能ですが、通常はパートのためにのみ使用されています。

集約の種類

属性のタイプがクラスの場合、この属性の値はオブジェクト、つまりクラスのインスタンスであることになります。この場合、属性には、属性を包含するクラスのインスタンスと値のインスタンスの間のライフタイム関係を決定する集約の種類を複数持たせることができます。

- **なし**

  2 つのクラスのインスタンスの間にライフタイム依存はありません。つまり、属性には、値クラスのインスタンスの参照が 1 つまたは複数あることになります。

- **共有集約**

  2 つのクラスのインスタンスの間にライフタイム依存はありません。しかし、非形式的には、一方がもう一方に「所有されている」と見なされます。属性入力領域では、共有集約は、shared a:myclass などのように、属性名の前に shared というキーワードを付けることで示されます。コード ジェネレータには、共有集約に特定のセマンティックを追加するものがありますが、実際には、セマンティックが弱いためにほとんど使用されません。通常は、集約のない関連を代わりに使用するほうがよいでしょう。

- **合成**

  包含するクラスのインスタンスと値クラスのインスタンスの間には、強力な一部／全体の関係があります。つまり、実際には、2 つのインスタンスの間にはライフタイム依存があるということになります。包含するインスタンスが終了すると、その中に包含されているインスタンスも終了します。合成は、part a:myclass などのように、属性名の前に part というキーワードを付けることで示されます。

注記

非静的属性には、定義コンテキストがインスタンス化された場合のみ値を持たせることができます。上述の、属性の定義コンテキストは、異なった方法でインスタンス化されます。たとえば、パッケージは使用時にインスタンス化され、イベント クラスは起動時にインスタンス化されます。

デフォルト値

属性には、式として指定したデフォルト値を持たせることができます。属性にデフォルト値がない場合は、値は、明示的に割り当てないかぎり、定義コンテキストのインスタンス化時に定義されません。

ポート

タイプがクラスの属性には、コネクタを接続できる通信ポートを持たせることができます。これらのコネクタは、シグナルを属性との間でやりとりする、システム内の通信経路を記述します。これは、属性がパートを表す場合に主に使用されます。

多重度

属性には、範囲の集合としてモデリングした多重度を持たせることができます。多重度は、属性がランタイム時に保持できるインスタンスの数を制限します。

属性の多重度が >1 であるかどうかにより、属性の実際のタイプが異なります。多重度が >1 の場合は、属性は値のリストを保持できるコンテナ タイプになります。多重度がちょうど 1（または `0..1`）の場合はそうなりません。

使用できるデータ型 ライブラリによって、コンテナ タイプも異なります。通常、コード ジェネレータが異なると、ターゲット言語と適切に統合されるよう、コンテナ タイプも異なります。特定のデータ型 ライブラリがロードされていない場合、付属の定義済みパッケージに、多重度 >1 の属性のタイプとして文字列タイプが使用されます。（文字列タイプは、整列リストまたはシーケンスを表す定義済みの集合 タイプです。リストの値は、属性のタイプに従わなければなりません。）

多重度は、クラス シンボルの属性入力領域で、次のように、属性のタイプの後に角かっこで囲んで示されます。

```
a :myClass [*]
```

上の例では、多重度は未接続です（アスタリスクで示します）。

つまり、値はいくつでも持つことができます。多重度が指定されていない場合、デフォルトで 1 と見なされます。

開始基数

多重度が >1 の合成属性の場合、式を使用してインスタンスの最初の数字を指定する省略表現があります。その数字により、所有側クラスのインスタンス化時に自動的に生成されるインスタンスの数を指定できます。インスタンスの最初の数字を省略すると、インスタンスは 1 つだけ生成されます。

注記

基数を解釈するかどうか、またその解釈の方法は、コード ジェネレータによって異なります。コード ジェネレータによっては、属性の基数を無視するものがあります。

属性が合成構造図でパート シンボルを使用して示されている場合は、開始基数を指定できます。ただし、このようなシンボルでは、構文は次のようになります。

```
a :myClass [*] / 2
```

ここで、a のインスタンスの開始基数は 2 です。

可視性

属性に可視性を指定できます。可視性は、**public**、**private**、**protected**、**package** のいずれかになります。

導出

属性を derived と宣言できます。これは、属性の値が対応するオブジェクトに格納されるのではなく、他の属性の値などから計算されることを示します。導出属性の構文では、次に示すように、属性名の前に / を付けます。

```
/a:myClass
```

導出属性のための導出規則の指定方法については、Derived（導出）を参照してください。

静的

静的属性は、インスタンス スコープでなくクラス スコープに所有されている属性です。これは、特定クラスのすべてのインスタンスに共有されている属性インスタンスが 1 つだけであるということです。

定数

定数属性は、値を動的に変更できない属性です。定数の値は、属性のデフォルト値です。

外部定数属性とは、値がモデル外で定義されるか、後で（ビルド タイムなど）定義されることを示します。

例 16: テキストによる定数宣言

```
const Integer a = 10;
const Integer extern ext_const;
```

操作

操作は、クラスのインスタンスが、操作のシグニチャと一致する呼び出しを処理できるという宣言です。操作は、操作本体または状態機械で実装できます。この実装（メソッドと呼ばれることが多い）は、操作が起動すると実行されます。これは、受信側がパッシブ インスタンスであれば、操作起動後直ちに実装が実行され、受信側がアクティブ インスタンスであれば、実装の実行は遅れ、後でインスタンスが操作コールを受け付けられる状態になったときに実行される場合があるということです。

操作は、クラス シンボルの操作入力領域で、および特殊操作シンボルを使用して、テキストによって宣言できます。

DOORS Analyst は、操作の `derived` プロパティを標準 UML の拡張機能としてサポートしています。このプロパティを使用して、操作は実装を伴わないが暗黙的に計算されることを示すことができます。このプロパティは、分析専用です。生成されたコードには影響しません。

シンボル

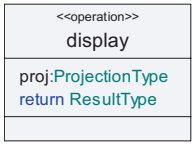

図 31: 操作

構文

シンボルには、操作ヘッダーとパラメータという 2 つの編集可能なテキスト フィールドがあります。下のフィールドは常に空白です。

アクティブ クラス

アクティブ クラスは、それ自体の制御スレッドがあるクラスです。アクティブというプロパティにより、通常のクラスと区別されます。描画的には、182 ページの図 32 に示すように、特殊なアクティブ クラス シンボルによって示されます。

シンボル

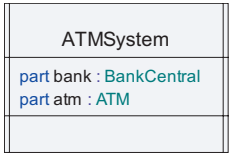

図 32: アクティブ クラス

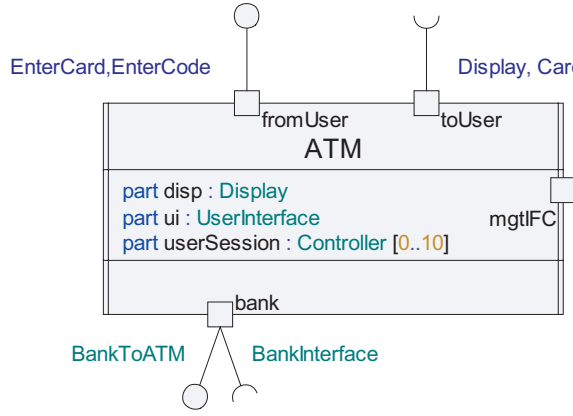

図 33: アクティブ クラスとポート、実現化インターフェイス、要求インターフェイス

クラスは、以下の方法によってアクティブにできます。

- クラスを選択し、ショートカット メニューの［アクティブ］を選択する
- クラスを選択し、プロパティ エディタで［アクティブ］を選択する

アクティブ クラスは、UML でリアルタイムの振る舞いをモデリングするための基本構成単位です。アクティブ クラスにより、モデルの構造（アーキテクチャ）と振る舞いの両方が定義されます。UML のアクティブ クラス 概念のこの二元性により、強力で柔軟な設計能力が得られます。

構造

アクティブ クラスの構造は、1 つまたは複数の合成構造図で定義され、アクティブ クラスは他のアクティブ クラスのインスタンスの集合として定義されます。これらのアクティブ クラスには構造を持たせることもでき、これで複雑なアーキテクチャの記述が可能になります。

振る舞い

アクティブ クラスの振る舞いは、1 つまたは複数の状態機械図の状態機械によって定義されます。状態機械には、`initialize()` と命名するか、クラスと同じ名前を付ける必要があります。

アクティブ クラスの仕様を完全に決定するには、構造定義か状態機械定義、またはその両方を持たせる必要があります。

アクティブ クラスにはそれ自体の制御フローがあり、振る舞いを開始することも、インターフェイスで見られる振る舞いにパッシブに反応することもできます。伝統主義者は、アクティブ クラスよりも *リアクティブ*クラスという名前を好みます。これは、

通常このようなクラスはイベント駆動であるためです。振る舞いは、通常、タイマーを使用して開始されます。タイマーがタイムアウトすると、何らかの振る舞いが起動します。

アクティブ クラスに、合成構造図で定義された、包含されているパートがいくつかある場合、各パートは非同期にシステムの他のパートと並行して実行されます。このセマンティックにより、モデルが分散物理環境に配置でき、共有メモリ アクセスによる単一プロセッサでの実行に依存しないようにできます。

アクティブ クラスは、ポートによってインターフェイスの実現化と要求ができます。ポートは、要求インターフェイスや実現化インターフェイスとともに、アクティブ クラスとその環境の間の静的規約を定義します。

属性と操作

アクティブ クラス シンボルでは、クラスの属性をシンボルの第 2 入力領域で、操作を第 3 入力領域で指定または表示できます。

注記

public 属性と操作がクラス シンボルに示されていても、クラスの外からアクセスできるようにするには、クラスで実現化されたインターフェイスの一部として宣言する必要があります。このため、アクティブ クラスの属性と操作の入力領域の使用は実用上あまり便利でなく、主として、インターフェイスの仕様が完全に決定される前の分析フェーズで使用されます。

参照

属性

操作

ポート

ポートは、アクティブ クラスの相互作用点で名前のあるものです。ポートにより、実装されたインターフェイス（実現化インターフェイス）と、必要な別のクラスのインターフェイス（要求インターフェイス）が指定されます。

ポートは通常、アクティブ クラスのみで使用します。作成済みのポートをアクティブ クラス シンボルまたはパート シンボルで可視化するには、ショートカット メニューの［表示／非表示］から［ポートの表示］コマンドを選択します。

シンボル

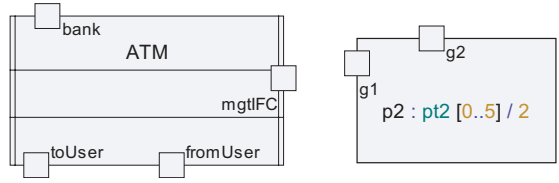

図 34: クラスのポートとパートのポート

シンボルには、名前の入るテキスト フィールドが 1 つあります。

ヒント

ポート シンボルを追加する最も簡単な方法としては、まずポート シンボルを置くシンボルのフレームを選択し、ツールバーのポート シンボルをクリックします。

ポートタイプ

シンボルには、名前の入るテキスト フィールドが 1 つあります。このフィールドには任意に **タイプ**を入れることができます。ポートのタイプは、主に分析フェーズで使用するために入力します。

注記

DOORS Analyst のコード ジェネレータではポート タイプを考慮しません。その代わりに、ポートの実現化インターフェイスと要求インターフェイス用に与えられた情報に基づいてコードが生成されます。

振る舞いポート

ポートには、振る舞いポートと非振る舞いポートの 2 種類があります。これらの違いは、振る舞いポートがクラスの状態機械と直接関連付けられているのに対し、非振る舞いポートはコネクタを使用して接続する必要があり、通常はクラス外からの通信をクラスの内部パートの一部に中継するだけのものであるという点です。

振る舞いポートは、クラスの状態機械と直接接続されているポートです。このポートに送られるシグナルはすべてクラス自体の振る舞いによって処理されます。

ポートとインターフェイス

ポートごとに、実現化インターフェイスと要求インターフェイスを指定できます。ポートの実現化インターフェイスにより、ポートを通じて処理できる着信要求が定義されます。要求インターフェイスにより、1 つまたは複数のコネクタを通じ外部からポートに接続されたクラスが処理しなければならない発信要求が定義されます。183 ページの図 33 に、実現化インターフェイスと要求インターフェイスのあるポートの例を示します。

アクティブ クラスの構造または振る舞いを定義する際、ポートは、この目的で使用されるダイアグラム（合成構造図または状態機械図）の境界で宣言できます。ポートはパートからも参照できます。この場合、パート シンボルの境界に示されます。

また、アドレッシング メカニズムとして、（追加されたコネクタのもう一方の端にある受信側に関する知識なく）ポートを通じて、状態機械からメッセージを送ることもできます。

ポートの実現化インターフェイス（または要求インターフェイス）には、通常、インターフェイスの参照に加えて、シグナル リスト、シグナル、属性の参照が含まれています。

ポートの実現化インターフェイスと要求インターフェイスは、実現化インターフェイス シンボルと要求インターフェイス シンボルをポートに追加することによって可視化されます。これらのシンボルでは、サポートされているインターフェイスや必要なインターフェイスの名前（またはシグナル リスト、シグナル、属性）が指定できます。

実現化インターフェイスや要求インターフェイスを指定する方法として他には、［プロパティ］ダイアログによるものがあります。

ポートによって、以下のものが表されます。

- インターフェイスとクラスの接続点
- これらのクラスを別のインスタンス、または囲む側のフレーム シンボルと接続する、合成構造図の接続ラインの接続点

ポート シンボルは、以下の場所に使用できます。

- クラス シンボル
- パート シンボル
- 振る舞いシンボル
- アクティブ クラスが所有している状態機械のフレーム
- 合成構造図のフレーム
- アーキテクチャ図や状態機械図の中（ポートがこれらの図のフレームに置かれた場合と同じセマンティックを持つ）

ポートには、explicit コネクタも implicit コネクタも持たせることができます。各ポート シンボルに追加できるインターフェイス シンボルの数は、0、1、2 のいずれかです。

インターフェイス シンボルが 2 つある場合、一方を、ポートへの着信インターフェイス（またはシグナル）を指定する実現化インターフェイス シンボルとして定義し、もう一方を、ポートからの発信インターフェイスを指定する要求インターフェイス シンボルとして定義しなければなりません。

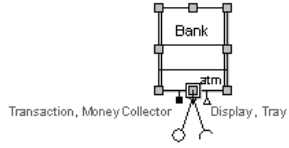

図 35: 実現化インターフェイスと要求インターフェイスのあるポート

インターフェイスのある、または、インターフェイスのないポートは、以下のいずれかの方法で、直接クラスに描画できます。

- クラスを選択して、Shift キーを押しながらツールバーの［ポート］シンボルをクリックします。新しいポート名を入力します。ポートはクラスの左枠の上左隅に近い位置に配置されます。
- ツールバーの［ポート］シンボルをクリックし、ポートの配置先のクラスをクリックします。名前テキスト フィールドを編集します。

継承

スーパータイプに所属するポートのあるクラス間の汎化の場合、これらのポートも継承されます。

ポートは、public または private と宣言することにより、ポートが外部に露出されるか、内部のみで使用されるかを区別できます。サブクラスのポートにシグナルを追加できます。

インターフェイス

インターフェイスは、インスタンス化できない構造化分類子です。インターフェイスを実装するクラスが実装する必要のある一連の属性、操作、シグナルをグループ化するために使用します。クラスがインターフェイスを実装することを、インターフェイスを**実現化**する、と言います。これで、インターフェイスで宣言された操作がサポートされます。クラスがインターフェイスを**要求**することもできます。この場合、クラスは、操作を実行するために他のアクティブ クラスに依存します。

シンボル

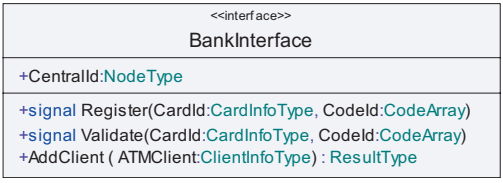

図 36: インターフェイス シンボル

インターフェイスの操作は、通常、インターフェイスを実現化するクラスによって提供されるサービスを記述します。当然、クラスは複数のインターフェイスを実現化できます。

操作の他に、インターフェイスにはシグナルや属性を持たせることもできます。

インターフェイスは特化が可能で、テンプレート パラメータを持たせることができます。インターフェイスの多重継承は、アクティブ クラスの通信インターフェイスの定義に便利な仕組みです。

インターフェイスは、関連インターフェイスを実現化するクラス間のプロトコルや規定を定義するために、互いに関連付けることもできます。MgmI インターフェイスと MgmReplyI インターフェイスを定義する例を 188 ページの図 37 に示します。2 つのインターフェイスを関連付けることにより、インターフェイス間に関係ができます。これで、一方のインターフェイスがポートなどで参照されたり、コネクタと関連付けられたりすると、もう一方のインターフェイスが自動的に反対方向に挿入されるようになります。したがって、あるポートを通じてクラスが MgmI インターフェイスを実現化した場合、MgmReplyI インターフェイスは自動的に同じポートの要求インターフェイスになります。

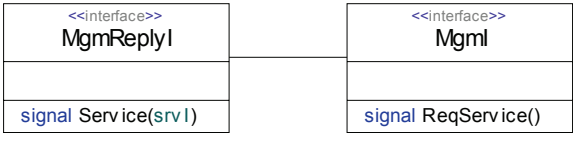

図 37: 2 つの関連インターフェイスを使用して定義された規定

構文

シンボルには、以下に示すように、編集可能なテキスト フィールドが 3 つあります。

- ヘッダー
- 属性

- 操作

ヘッダー フィールドは、インターフェイスの名前の定義に使用します。

属性フィールドには、インターフェイスを実現化するクラスが実装しなければならない属性の定義が入ります。通常、これは、実現化するクラスの protected 属性への getter 操作と setter 操作の省略表現です。

操作フィールドには、インターフェイスを実現化するクラスが処理しなければならない操作とシグナルの定義が入ります。

参照

実現化インターフェイス

要求インターフェイス

実現化インターフェイス

クラスのポートに追加された実現化インターフェイスは、クラスがそのポートによって実現化するインターフェイスを可視化します。インターフェイス、シグナル、シグナル リスト、属性をテキスト フィールドで指定できます。

シンボル

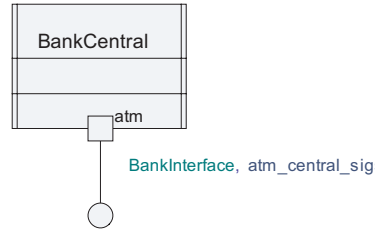

図 38: 実現化インターフェイス

構文

シンボルには、テキスト フィールドが 1 つあります。

例 17: 実現化インターフェイス

```
S, p, SigList
```

参照

インターフェイス

要求インターフェイス

要求インターフェイス

クラスのポートに追加された要求インターフェイスは、クラスがそのポートによって処理されるものと見なす要求を可視化します。インターフェイス、シグナル、シグナル リスト、属性をテキスト フィールドで指定できます。

シンボル

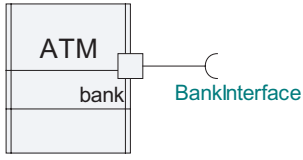

図 39: 要求インターフェイス

構文

シンボルには、テキスト フィールドが 1 つあります。

例 18: 要求インターフェイス

```
S, p, SigList
```

参照

インターフェイス
実現化インターフェイス

シグナル

シグナルは、UML における通信の主要手段の 1 つです。シグナルは、アクティブ クラス間で送信される非同期メッセージを表します。シグナルにより、データを伝送できます。データは、シグナルの宣言パラメータ型に一致しなければなりません。

シグナルの最も便利な宣言の方法として、インターフェイスを実現化するクラスの能力を表すインターフェイスで他のシグナル、操作、属性とともに宣言します。

ただし、191 ページの図 40 に示すように、クラス シンボルに類似した特殊シグナル シンボルの使用により、スタンドアロン シグナル宣言も可能です。

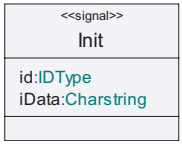

図 40 シグナル

異なるシグナルを多数使用する場合は、テキスト シンボルを使用してシグナルをテキストで宣言したほうが実用的なことが多いでしょう。

例 19: テキストによるシグナル宣言

```
signal Init (IDType id, Charstring iData);
signal SetupReq, SetupInd, AbortReq, AbortInd;
signal ForwardedMsg (IDType, MsgData);
```

構文

シグナル シンボルには、以下に示すように、編集可能なテキスト フィールドが 2 つあります。

- ヘッダー
- パラメータ

ヘッダー フィールドではシグナルの名前を宣言し、パラメータフィールドではシグナルのパラメータを宣言します。パラメータの名前は省略してもかまいませんが、パラメータ タイプは必須です。

多くのクラス的シンボルに設けられている 3 つ目の入力領域は、シグナル シンボルの場合は常に空白です。

参照

メッセージ
シグナル リスト
インターフェイス
タイマー

シグナル リスト

`signallist` キーワードは、記述をわかりやすくするために、関連するシグナルのグループを表すために使用されます。通常、ポートやコネクタに使用されます。

例 20: シグナル リスト宣言

```
signallist MgtSignals = MOGetStatus, MOSet, MOReset;
```

注記

インターフェイスを使用したシグナルのグループ化は、シグナル リストよりも構造化されたアプローチです。これは、インターフェイスがシグナル宣言をカプセル化するためです。

参照

シグナル

インターフェイス

タイマー

タイマーは、シグナルと同様に、遷移のトリガとなるイベントです。タイマーはアクティブクラスが実行している実装コードによって設定され、タイムアウト時に、タイマー イベントを同じアクティブクラスインスタンスの状態機械で受信できます。時間値はアクティブ タイマーに関連付けられ、タイムアウトの時間となります。

シンボル

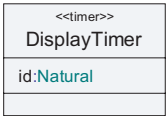

図 41: タイマー

シグナルと同様、タイマーにもパラメータを持たせることができます。これを利用して、すでにアクティブになっているタイマーをリセットせずに、同じ種類のタイマーを複数設定できるようにできます。つまり、パラメータの異なる複数のタイマーを同時にアクティブにできます。

構文

タイマーは、次に示すように、テキスト シンボルを使用してテキストで宣言することもできます。

例 21: テキストによるタイマー宣言

```
timer DisplayTimer (Natural id) = 2;
timer BankTimer () = BankTimeout;
timer UserTimer ();
```

タイマーをテキストで宣言する場合、タイマーにデフォルト持続時間、つまり、タイムアウトまでの時間を与えることもできます。これで、時間を指定せずにタイマーを設定できます。

参照

タイマー設定アクション

タイマー リセット アクション

タイマー設定

タイマー リセット

タイマー タイムアウト

データ型

データ型は、以下の 2 つ目的で使用されます。

- 使用できる基本タイプの記述
- ユーザー定義列挙タイプの記述

基本タイプは、多くの場合、特定の UML プロファイル（スタンドアロン プロファイルまたは特定のコード ジェネレータとともに使用するために定義されたプロファイル）に付随するモデル ライブラリで定義されます。後者の場合は、通常、データ型によってターゲット言語の基本タイプが定義され、UML モデルで使用できるようになります。

基本データ型をユーザー モデルで定義することもできますが、コード生成に問題が生じる場合があります。

列挙は、単にリテラルのリストとして値を列挙することによって一連の値を定義します。

いずれの場合も、データ型には、**操作**によって定義された振る舞いを任意で持たせることもできます。

シンボル

図 42: 列挙データ型

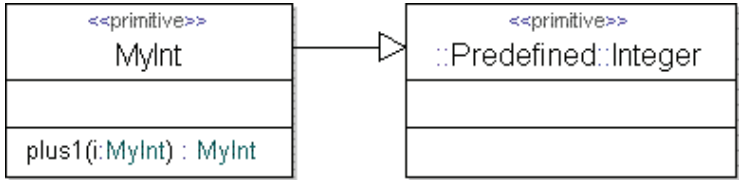

図 43: 演算子のあるデータ型

列挙データ型

列挙データ型は、リテラル値が論理名であるようなデータ型です。論理名は、オプションで、単純式で指定された複合的な値に追加できます。

使用できるデフォルト演算は以下のとおりです。

- 等価（==, !=）
- 関係演算（<, >, <=, >=）
- 代入（=）

例 22: 列挙データ型

```
enum UKColors { blue, red, white }

enum LinePrinterState {
  outOfService = 1,
  inServiceFree =2,
  inServiceBusy = 6
}

void op() {
  LinePrinterState e;
  Integer i;
  e = cast<LinePrinterState>(1);
  e = inServiceFree;
        i = cast<Integer>(e);
}
```

注記

194 ページの例 22 の操作 op のように cast 演算を使用して、整数と列挙タイプの間で変換を行うことができます。

基本データ型

基本データ型は通常、プロファイルのモデル ライブラリで定義されますが、ユーザー定義も可能です。ただし、ユーザー定義の基本タイプにはリテラル構文がありません。したがって、実際にはあまり有用ではありません。

データ型を別の既存データ型と関係付ける方法には以下の 2 つがあります。

- コピー コンストラクタを使用する
- 継承を使用する

いずれの場合も、既存データ型のリテラル構文が使用されます。UML に新規基本データ型を導入する方法として、コピー コンストラクタ メカニズムを推奨します。ほとんどのモデル ライブラリで使用されているのがこの仕組みです。

注記

基本データ型は通常、コード ジェネレータで特別な処理が必要です。ユーザー定義基本データ型は、コード ジェネレータのドキュメントで特に言及されていなければ、コード ジェネレータでは機能しません。

例 23: 演算子のあるデータ型

```
datatype simpleInt {
  simpleInt(Integer) {}
}
datatype myInt : Integer
{
  myInt plus1 ( myInt i) { return i+1;}
}
```

リテラル

リテラルは、列挙されたデータ型によって定義されたタイプの要素です。リテラルはそのデータ型に所有されます。リテラルの可視性は常に public です。

リテラルには、名前に加えて（名前はすべての定義にあります）、算術式での使用を可能にする整数値を持たせることができます。

選択

選択は、1 つの値を保持できるデータ型です。この値は、実行時はデータ型が複数でもかまいません。タイプの選択は、変数に値を割り当てる際に行われます。タイプ フィールド候補ごとに、フィールドの有無を確認する論理演算子 `IsPresent()` があります。

例 24: 選択

```
choice IntOrBool {
    Integer a;
    Boolean b;
}

IntorBool ib;
Integer i;
Boolean b=true;

ib.a=5;
i=ib.IsPresent("a")?ib.a:0; /* check if ib is Integer;
       if Integer, return ib,
       if not, return 0 */
ib.b=b;
```

例 25: 選択

```
choice IntOrBool {
  Integer a;
  Real r;
  Integer GetInt() {
    if (IsPresent("r")) {
      return 0;
    } else {
      return a;
    }
  }
}
```

IsPresent() 演算子の使用例

```
IntOrBool MyVar;
Real num_real;
Integer num_int;
MyVar.a=1;
if(IsPresent(MyVar,"a"))
  {
    num_int =MyVar.a;
    MyVar.r=3.14;
  }
  else
  {
    num_real=MyVar.r;
  }
if (MyVar.IsPresent("r")) {
switch (MyVar.r) {
  case 3.14 :
  {
    nextstate idle;
  }
  default :
  {
    nextstate idle;
  }
}
```

選択インスタンス値は、1 つの代入値（choice_field = value）のみを持つインスタンス式で指定できます。

例 26: 選択インスタンスの値

```
choice choice_type
{
  public Integer ifield;
  public Boolean bfield;
}
choice_type an_int = choice_type (. ifield = 1.);
```

シンタイプ

シンタイプは、別のデータ型（親 タイプ）に基づくデータ型です。2 つのタイプは、タイプ互換性とリテラルの点では同じです。シンタイプのリテラルは、親のリテラルと同一か、親のリテラルのサブセットです。シンタイプは、別のタイプのエイリアス（制約を付けることのできる）と見なすことができます。

例 27: シンタイプ

```
syntype myInt = Integer constants (> -10, != 0, <10);
syntype smallPrime = Natural constants (1,2,3,5,7);

Integer [1..10] myvar; /* inline syntype definition */
```

状態機械

状態機械の概念は、振る舞いモデリングセクションで詳しく説明しています。

ステレオタイプ

ステレオタイプの概念は、拡張性セクションで詳しく説明しています。

関係

クラス図では、以下の関係を使用できます。これらの詳細については、UML の関係セクションを参照してください。

- 関連
- 集約
- 合成
- 依存
- 拡張
- 汎化
- 実現化
- 表現

オブジェクトモデリング

クラスモデリングが設計しているアプリケーション中のオブジェクトの種類に対して焦点を当てているのに対して、オブジェクトモデリングはオブジェクトがどのように実行時に現れるのかについて関心を払うモデリングです。この分析作業で発せられる典型的な疑問は以下のようなものになるでしょう：

- **異なる時点でアプリケーションに存在するオブジェクトは何か？**
- **オブジェクトは属性値によってどのような見え方をするか？**

- **オブジェクト同士はどのように関連付けられるか？あるオブジェクトについての知識をもっているのはどのオブジェクトか？**

オブジェクトはインスタンスとも呼ばれます。したがって、この分析作業をインスタンスモデリングと呼ぶこともあります。

通常オブジェクトモデリングとクラスモデリングは並行に行われます。アプリケーションのオブジェクトが識別されると、それらはモデルの中で定義されます。この作業は、オブジェクトの種別を見つけ出す以前に行われることもあります。

実際のアプリケーションでは実行時に現れるオブジェクトの数はきわめて大きくなります。したがって、設計上注目すべきオブジェクトのみをモデルとして描いてゆく、というアプローチを取ること多くなります。たとえば、アプリケーションの初期化時の動作を理解するという目的のためには、そのアプリケーションの起動時に作成されるオブジェクトのみを検出してゆくというやり方になります。

オブジェクトモデリングでは、主として オブジェクト図を使用して、オブジェクトとその関係を定義してゆきます。同時にクラス図を使ってゆく場合もあります。

オブジェクト図

オブジェクト図は、特定の時点でアプリケーションに存在するオブジェクトのビュー（いわゆる「スナップショット」）を提供します。オブジェクト図に現れるオブジェクトは、名前が与えられ、場合によっては型（種別）が決まることもあります。「スロット（Slot）」と呼ばれるオブジェクトの属性値を与えることもできます。オブジェクト間の連結関係は、リンクラインで示されます。

名前の付けられたオブジェクトは、Tau では、**名前付きインスタンス**と呼ばれて、他の名前のないインスタンス、たとえば適用済みステレオタイプインスタンスなど、とは区別されます。名前付きインスタンス の定義は、デフォルトでオブジェクト図を含むスコープに配置されます。ただし、名前付きインスタンスを、［モデルビュー］からオブジェクト図にドラッグアンドドロップして、別のスコープで表示することも可能です。

オブジェクト図は、1 つの名前付きインスタンスを表している複数のインスタンスを含んでいる場合もあります。

オブジェクト図の例

下図のオブジェクト図は、168 ページの図 28 のクラス図で説明したアプリケーションで使用可能なオブジェクトの、スナップショットビューを示しています。

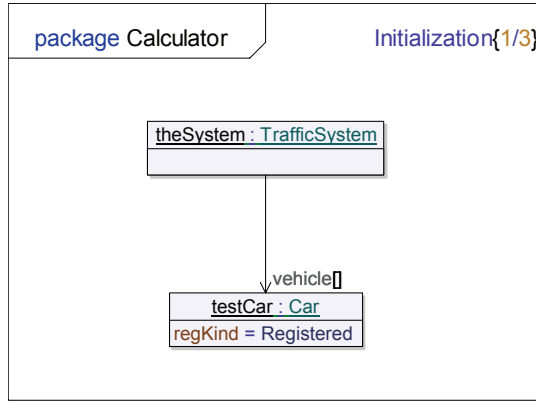

図 44: オブジェクト図

このオブジェクト図は、ある時点（ダイアグラム名から判断するにおそらく初期化時）でアプリケーションが TrafficSystem クラスのインスタンスを一つ含むということを、示しています。そのインスタンスは、vehicle リスト属性内の testCar と呼ばれる Car のインスタンスを保持します。testCar オブジェクトの regKind 属性は、値 Registered を持っています。

オブジェクト図のモデル要素

オブジェクト図では、以下のモデル要素を表現できます :

- 名前付きインスタンス
- スロット（Slot）
- 依存

参照

クラス図 .

名前付きインスタンス

名前付きインスタンスは、モデル化されたシステム内のオブジェクト（インスタンス）表し、そのオブジェクトについての完全または部分的な説明を付与します。オブジェクトは時とともに変化するので、名前付きインスタンスが提供するのは特定時点または特定の期間でのオブジェクトの情報です。UML オブジェクト図では、以下の点について形式的な表現ができないことに注意してください。

- オブジェクトと名前付きインスタンスとが合致する時点、期間

- 名前付きインスタンスがオブジェクトの完全な仕様を含むか、部分的な仕様しか含まないか

通常、名前付きインスタンスには名前があります。多くの場合子の名前は非形式的に解釈され、名前付きインスタンスで記述される実行時オブジェクトのどのプロパティにも対応しません。ただし、定義についての一般的な規則には従う必要があります。たとえば、同じスコープにある複数の名前付きインスタンスの名前は一意である必要があります。(スコープ、モデル要素、ダイアグラム参照)

通常、名前付きインスタンスには型があります。指定された型がクラスの場合は、名前付きインスタンスはそのクラスのオブジェクトを記述しています。指定された型がデータ型の場合は、名前付きインスタンスはそのデータ型の値を記述しています。操作、シグナルなどの振る舞い特性を型として指定することもできます。その場合、名前付きインスタンスはシステム内のイベントを記述することになります。たとえば、型が操作の場合は、名前付きインスタンスは操作呼び出しを記述し、型がシグナルの場合は、名前付きインスタンスそのシグナルのイベントを記述します。

名前付きインスタンスの型として、関連を使うこともできます。その場合、名前付きインスタンスは、リンクを表現します。

名前付きインスタンスについて抽象型を指定することもできます。これは記述される対象のオブジェクトが抽象オブジェクトであるということではなく、オブジェクトの表示されるプロパティがすべて抽象型のみであることを意味します。記述される実行時オブジェクトは、その抽象型の具体的なサブタイプになります。

名前付きインスタンス型が、クラス属性やシグナルパラメータのようなストラクチャフィーチャを持っている場合、名前付きインスタンスはそのストラクチャフィーチャに対して値を指定します。このような値の指定をスロット（Slot）と呼びます。

名前付きインスタンスは、オブジェクト図では **InstanceSymbol**. を使用して表示されます。

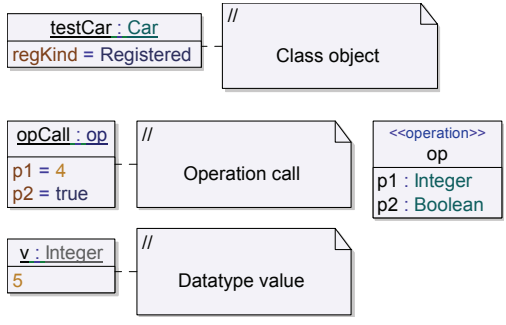

図 45: 名前付きインスタンスを定義しているインスタンスシンボル

上図のように、インスタンスシンボルは二つの基本区画を持っています。上の区画は、名前付きインスタンスの名前と型を保持します。下の区画はスロットを保持します。スロットの定義の構文は通常の割り当て文の構文と同じであることに注意してください（ストラクチャフィーチャは割り当てられた値です）。データ型値には、単純な値も使用できます。

注記

現在のセマンティックチェッカはデータ型値とデータ型の間の型互換性をチェックしません。したがって、オブジェクト図中のデータ型値は、非形式的なモデリング向けです。

リンク

リンクは、型が関連である名前付きインスタンスです。リンクは 2 つのオブジェクトの間の実行時の関係を図示します。プログラミング言語の言葉で言えば、リンクはポインタまたは参照に対応するといえるでしょう。

リンクはオブジェクト図では以下の 2 つの方法で表示されます :

1. 2 つのインスタンスシンボルを連結するリンクライン。
2. インスタンスシンボル内の通常のスロット。スロットの右手がターゲットの名前付きインスタンスを指します。

図 46: 2 通りのリンクの指定法

リンクラインのターゲット端に入力できるテキストは、式です。このテキストは、スロット（Slot）式の左辺です。

リンクの名前は、リンクラインの中央にあるラベルに入力して指定できます。

スロット（Slot）

スロットは、名前付きインスタンスの型に属するストラクチャフィーチャに対して値を指定する場所です。

スロットには、ある一般的なオブジェクトの値を示すという用途があります。名前付きインスタンスにスロットが定義されていないということは、必ずしも、対応するオブジェクトに何もストラクチャフィーチャがないということを意味しません。単に、その値がモデリング上興味の対象になっていない、ということにすぎません。

スロットは、ある特定の型のすべての種類のストラクチャフィーチャ（継承したフィーチャや public ではないフィーチャも含む）を参照する可能性があります。

スロットは、ストラクチャフィーチャ（左辺）に対する値（右辺）の割り当てになります。右辺は、単純な識別子であることが多いですが、より複雑な式を記述することもあります。以下のモデルを参照してください：

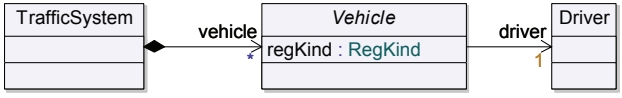

図 47: 関係のある 3 つのクラス

TrafficSystem の 1 つのインスタンスに定義されたスロットは、この例では、以下の表にあげた左辺をもつことができます。

スロットの左辺	意味
vehicle[]	`vehicle` コレクションの中の 1 つのインスタンス。コレクションのインデックス値は指定されていません。
vehicle[4]	`vehicle` コレクションの中の 1 つのインスタンスで、印でクス値 4 が指定されているもの。
vehicle[].driver	`vehicle` コレクションの中の 1 つのインスタンスの `driver` インスタンス。

最後の例がリンクラインとともに可視化されると、間接的なリンクはコンパクトな表記になります：

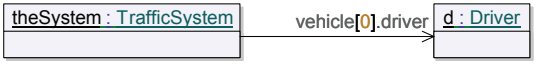

図 48: traffic system から最初の vehicle の driver へのリンクを可視化

自己参照

オブジェクトの自己参照の指定には、同等な 2 つの表記があります。スロットの右辺は、包含している名前付きインスタンスへの参照か、または、this キーワードです。

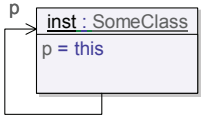

図 49: リンクラインのある自己参照のスロットラベル

アーキテクチャ モデリング

アーキテクチャ モデリングでは、アクティブ クラスの内部構造が、通信の観点から記述されます。これは、クラスの属性（この状況ではパートと呼ばれます）をコネクタと接続し、これらのコネクタで送ることのできるシグナルを指定することによって行います。このパートとコネクタの構造を、クラスのアーキテクチャまたはクラスの合成構造と呼びます。

アーキテクチャ モデリングは通常、設計フェーズでクラス モデリングと並行で、またはクラス モデリングの後に行います。

合成構造図

合成構造図（旧アーキテクチャ図）は、他のアクティブ クラスとの関連で、アクティブ クラスの内部ランタイム構造を定義します。これらの構成単位は、包含するクラスの合成パートである場合は、パートと呼ばれます。また、パートはアクティブ クラスのインスタンス化のみに限定されます。合成構造図で、パートの通信ポート間のコネクタを可視化することにより、アクティブ クラス内の通信を表現することもできます。

例

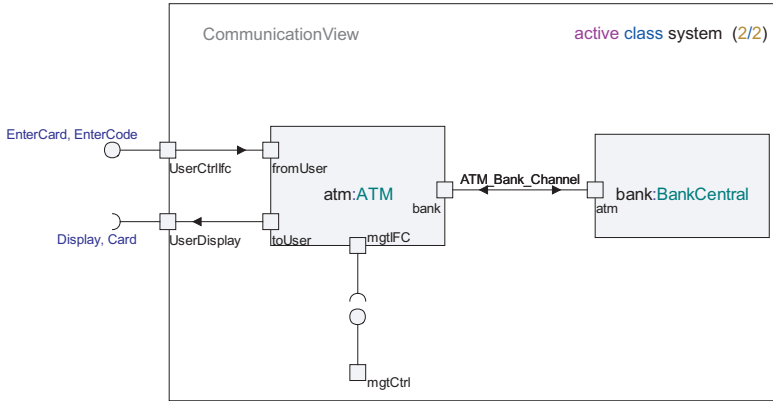

図 50: パート、ポート、コネクタを示す合成構造図

パート

パートは、包含するクラス インスタンスによって所有される 1 つまたは複数のインスタンスを表します。

すべての属性について、パートに多重度を持たせ、ランタイム インスタンスの数を制約できます。パートの多重度が >1 の場合、パートにはコンテナ タイプが仮定されます。コンテナ タイプは、ロードされているプロファイルやアドインによって異なりますが、デフォルトは文字列タイプが使用されます。

包含するクラスのインスタンスが生成されると、これらのパートに対応する一連のインスタンスを、直ちに、または後で、パートの開始基数と多重度で記述されたとおりに生成できます。

シンボル

図 51: パート

- パート シンボルに名前しかない場合、パート シンボルが生成されると、暗黙的クラスが自動的に構築されます。
- 同じ名前を持つ複数のパート シンボルを合成構造図に使用できます。

参照されているクラスを省略すると、インライン クラス定義のあるパート定義に相当します。このようにパートを指定すると、クラス定義がクラスの使用から切り離されず、記述がより簡潔になりますが、再利用にはあまり適さなくなります。

アクティブ クラスのパートは、アクティブ クラス シンボルの属性入力領域に表示できます。パートは包含するクラスの 1 つの属性となり得るからです。属性がパートである場合、その描画はコンテナ クラスとパート クラスの間の包含関係を記述しています。

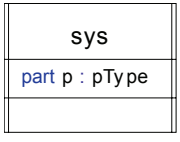

図 52: クラス シンボルの属性入力領域で可視化されたパート

また、205 ページの図 53 に示すように、クラス図で合成関係を使用してパートの階層の概要を表すこともできます。

開始基数によって、包含するエンティティの生成時に自動的に生成される開始インスタンスの数が決まります。開始基数を指定しない場合は、初期に生成されるインスタンスの数は、パートの多重度の下限と同じになります。多重度を指定しない場合は、自動的にインスタンスが 1 つ生成され、同時インスタンスの数に上限はなくなります。これらのインスタンスは、パートのタイプを決める分類子のインスタンスです。

パートは、ポートにコネクタを追加することにより結合できます。パートは、静的 / 動的に生成 / 終了されたアクティブ インスタンスの記述に使用します。

パートは、一連のインスタンスが存在し得ることを示します。この一連のインスタンスは、パートのタイプを決める分類子によって指定されたインスタンスのトータル セットのサブセットです。包含するクラスのインスタンスが終了すると、包含されているインスタンスも終了します。

パート シンボルは、モデルの属性を示します。合成構造図でのパート シンボルの外観は、対応する属性の集約の種類によって異なります。集約の種類が合成の場合、パート シンボルの輪郭は実線になります。集約の種類が参照または共有の場合、パート シンボルの輪郭は点線になります。

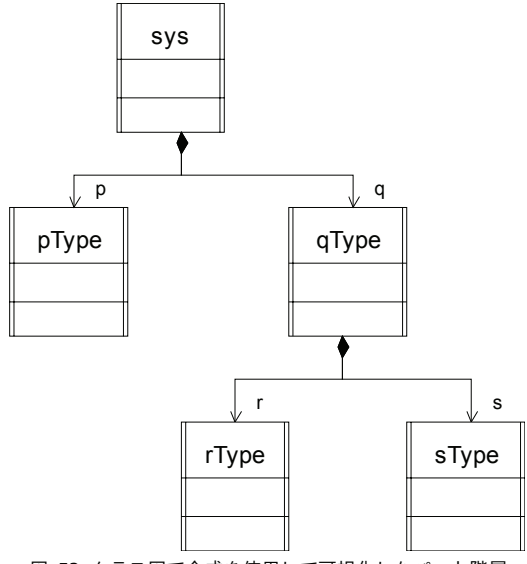

図 53: クラス図で合成を使用して可視化したパート階層

例 28: 単純なパート

```
myP
```

例 29: タイプ ベースのパート

```
myP :PT
```

例 30: 開始インスタンス数と最大インスタンス数を指定したパート

```
myP :PT [0..10] / 1
```

コネクタ

コネクタは、アクティブ クラスのパート間、またはアクティブ クラスの環境とそのパートの 1 つとの間の通信を可能にする手段です。コネクタにより、通信経路を直感的に可視化できます。

コネクタには一方向のものも双方向もあり、各方向で、許可される情報を指定します。コネクタによって送信または伝送可能な情報は、シグナル、属性、シグナル リスト、インターフェイスで記述できます。シグナルの数が多い場合は、コネクタの各方向に使用するインターフェイスやシグナル リストを定義したほうが便利です。

デフォルトで、コネクタに名前はなく、非遅延かつ双方向です。コネクタライン上のショートカット メニューから、コネクタ ラインのプロパティを制御できます。

シンボル

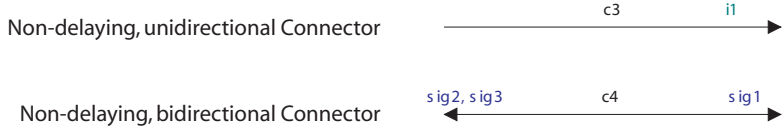

図 54: コネクタ タイプ

コネクタ ラインは、2 つのエンド ポイント、たとえばダイアグラムのパート シンボル、振る舞いシンボル、フレームなどに追加されたポート間の通信経路を指定します。

- 必要であれば、コネクタは省略してもかまいませんが、暗黙的に生成されます。
- コネクタは、ショートカット メニューから方向の変更や双方向化が可能です。
- 双方向コネクタの方向を変更すると、シグナル リスト領域が入れ替わります。
- コネクタの名前はオプションです。
- コネクタと関連付けられたインターフェイスやシグナルなどのリストはオプションです。

アクティブ クラスの構造には、explicit コネクタ ラインか implicit コネクタ ライン、またはその両方を入れることができます。explicit コネクタは可視ですが、implicit コネクタは不可視で、参照できません。

implicit コネクタは、以下のところにある実現化インターフェイスと要求インターフェイスで該当するすべてのものから計算されます。

- 包含するクラスに包含されたパートのポート
- 包含するクラスのポート
- 包含するクラスの振る舞いポート

注記

ポートに explicit コネクタがある場合は、そのポートには implicit コネクタは接続されません。

構文

ラインには、2 つ（一方向コネクタ）または 3 つ（双方向コネクタ）の編集可能テキストフィールドがあります。

中央のフィールドではコネクタの名前を指定し、ラインの終わりのフィールドではシグナル リスト領域を指定します。ラインの各矢印にシグナル リスト領域が 1 つあります。シグナル リスト領域は空白でもかまいません。

コネクタ ラインに適用されたステレオタイプは、名前フィールドの上にあるテキストフィールド（編集不可）に表示されます。

例 31: コネクタ シグナル リスト

```
i1,i2,sl1
```

シグナル リストとインターフェイス

シグナル リストからポートへのコネクタを描画できます。その場合、以下のようになります。

- シグナル リストにシグナルまたはインターフェイスがない場合、シグナルとインターフェイスの推定のため、接続したポートが使用されます。
- コネクタに関連付けられたシグナル リストにシグナルまたはインターフェイスもない場合、トランスポートされたすべてのシグナルとインターフェイスを指定する必要があります。

合成構造図のコネクタ ラインのショートカット メニューに［すべてのシグナルを表示］コマンドがあります。このコマンドで、シグナル リスト テキスト フィールドに、接続ポートから取得したシグナルとインターフェイスのリストを挿入できます。

- このコマンドの実行で変更されたリストから、既存のシグナルとインターフェイスを削除することはできません。
- 既存していないシグナルとインターフェイスのみ、シグナル リストに追加できます。
- 2 つの接続ポートにあるシグナルとインターフェイスを結合できます。これで、1 つのポートに表示されるシグナルまたは、インターフェイスを、シグナル リストに表示できます。

- シグナルが実現化または、要求された場合、どのシグナル リストにシグナルを挿入するか決定されます。

パート コミュニケーション

通常、パート間のコミュニケーションはポートとポート間のコネクタ ラインで明示的にモデル化されます。

パート間のコミュニケーションがあいまいでない場合、つまり、ダイアグラム内のパートのクラスに、唯一の手段で接続されるポートが定義されている場合、これを明示的にモデル化する必要はありません。

パート シンボルに直接、コネクタを接続できます。この場合のパート シンボルとコネクタの振る舞いは、名称未設定ポートが作成されてパートに接続され、コネクタがこのパートに接続されます。コネクタの作成時と、既存コネクタへの再接続時の両方で、この方法を活用できます。この名称未設定ポートは、コネクタ ラインが削除された場合に削除されません。このポートがモデルに必要ない場合は、手動で削除する必要があります。

振る舞いポート

アクティブ クラスに構造がある、つまりパートがある場合でも、それ自体の振る舞いを持たせることができ、これは状態機械として表現します。この振る舞いは、合成構造図で振る舞いポートを使用して参照できます。

振る舞いポートの主な目的は、アクティブ クラスのパートとアクティブ クラスの振る舞いの間のコネクタを定義する場合にあります。この場合、振る舞いポートが必要です。

状態機械の通信インターフェイスを定義するため、パートのポートの場合と同様に、コネクタを振る舞いポートに追加できます。1 つの図に複数の振る舞いポートを使用できます。この場合、ポートは同じ基礎振る舞いを参照します。

シンボル

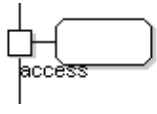

図 55: 振る舞いポート

振る舞いポート シンボルは、定義されたクラスの固有状態機械の参照を指定します。

- 1 つの図に複数の振る舞いポート シンボルを使用できます。
- このシンボルには、テキスト フィールドはありません。

振る舞いポートの外観は、クラス図の通常のポートと同じです。付加された振る舞い情報は、アーキテクチャ図と状態機械図にのみ表示されます。

ヒント

振る舞いシンボルを合成構造図に追加する方法は 2 つあります。ポート シンボルをダイアグラムに追加するか、[モデル ビュー] ブラウザから既存ポートを合成構造図にドラッグします。いずれの場合も、コマンド振る舞いポートをショートカット メニューから選択する必要があります。

関係

依存

合成構造図の依存関係は、パート間で使用され、あるパートが別のパートに依存していることを示します。一般的な使用方法として、パート間の生成依存、つまり、あるパートのインスタンスにより別のパートの新規インスタンスが生成できるということを示します。

コンポーネント モデリング

コンポーネント モデリングでは、システムの主要コンポーネントを特定し、そのインターフェイスと関係をモデリングします。

コンポーネント モデリング時の最重要点は、実装の詳細をコンポーネント内部に隠すことにより強力なカプセル化を実施し、明確に定義された少数のインターフェイスのみを露出することです。

コンポーネント間の弱い結合、つまり依存を最小限にすることも、コンポーネント モデリング時によく適用される設計原則です。

コンポーネント図

コンポーネント図では、一連のコンポーネント、その関係、および実現化インターフェイスと要求インターフェイスによってシステムの静的構造を記述します。クラスやアーティファクトなどの他のモデル要素もコンポーネント図に示して、コンポーネントとの関係を表すことができます。

例

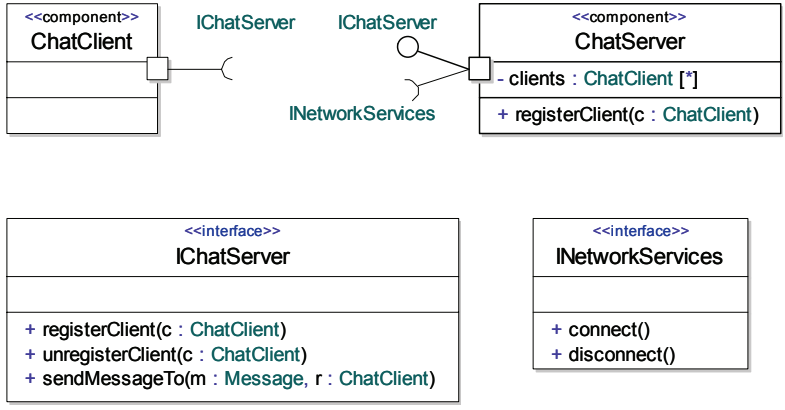

図 56: コンポーネント図

コンポーネント図のモデル要素

コンポーネント図には、以下の要素が使用されます。

- コンポーネント
- アーティファクト
- クラス
- インターフェイス
- ポート
- 実現化インターフェイス
- 要求インターフェイス
- 関係

参照

クラス図

コンポーネント

コンポーネントは、システムの小さいパートをカプセル化したもので、明確に指定されたサービスを提供します。

コンポーネントが提供するサービスは、その実現化インターフェイスによって指定されます。コンポーネントには、実現化インターフェイスによってのみアクセスします。コンポーネントは他のサービスに依存する場合もあります。これは、その要求インターフェイスによって指定されます。

コンポーネントの実装、すなわち振る舞いとアーキテクチャは、クライアントに露出されてはなりません。インターフェイスのみが露出された場合、クライアントに影響を及ぼさずに、コンポーネントを、まったく実装の異なる別のコンポーネントと簡単に置き換えることができます。

UML において、クラスとコンポーネントの違いはほとんどなく、相互に置き換えが可能です。クラスでできることはすべてコンポーネントでもできます。ただし、コンポーネントを使用する場合は、上述の設計原則に従う必要があります。

シンボル

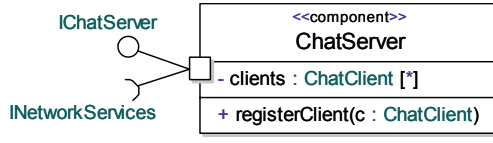

図 57: コンポーネントとポート、実現化インターフェイス、要求インターフェイス

コンポーネント シンボルは、クラス シンボルと同じです。ただし、<<component>> というキーワードを上部に追加します。

参照

クラス .

関係

コンポーネント図では、以下の関係を使用できます。

- 関連
- 集約
- 合成
- 依存
- 汎化
- 実現化
- 表現

アクティビティ モデリング

アクティビティ モデリングでは、アクティビティ図を使用して、振る舞いを小さい振る舞い単位に組織化することによりモデリングし、その単位間の制御とデータ フローを記述します。また、システムにおけるこれらの単位の分散も記述できます。

アクティビティ モデリングを抽象レベルでビジネス モデリングに使用したり、非常に低いレベルで使用してアクション コード レベルでの振る舞いをモデリングしたりすることもできます。非同期の分散システムのデザインに特に有効です。

参照

シナリオ モデリング

振る舞いモデリング

アクティビティ図

アクティビティ図では、振る舞いがどのように小さい振る舞い単位、アクション ノードに分割されるかを記述し、アクティビティ エッジ、および分岐ノード、フォーク ノード、アクティビティ終了 ノードなどの制御構成要素を使用して、単位間の実行シーケンスを制御します。

複数のアクション間でのオブジェクトとデータの受け渡しを記述するためにオブジェクト ノードとピンを使用します。

アクティビティ区画は、関連アクションを、たとえば機能や所有者ごとに関連グループにまとめるために使用します。

アクティビティ図はフローチャートに似ています。

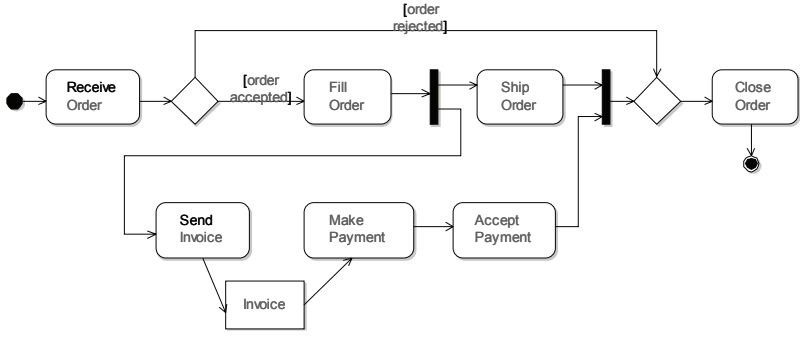

図 58: アクティビティ図

アクティビティ図の作成

アクティビティ図は、パッケージ、クラス、ユース ケース、操作、およびアクティビティに含めることができます。

1. ［モデル ビュー］でアクティビティ図を作成する場所となるエンティティを選択します。
2. ショートカット メニューから［新規］を選択し、［アクティビティ図］を選択します。

フロー方向

デフォルトでは［水平］に設定されています。

アクティビティ図で水平方向を選択すると、水平方向のアクティビティ フローを簡単に作成できます :

- ライン ハンドルはシンボルの外枠の右中央に配置されます。
- 新規の［フォーク／ジョイン］シンボルはデフォルトで垂直方向に設定されています。（デフォルトの方向が変更されても、既存の［フォーク／ジョイン］シンボルの設定は変更されません。）
- 新規の区画 シンボルのヘッダー サイズは、デフォルトで、高さが幅より大きく設定されています。（デフォルトの方向が変更されても、既存の区画 シンボルの設定は変更されません。）
- ライン ハンドルはシンボルの外枠の下中央に配置されます。
- 新規の［フォーク／ジョイン］シンボルはデフォルトで水平方向に設定されています。
- 新規の区画 シンボルのヘッダー サイズは、デフォルトで、幅が高さより大きく設定されています。

Shift + Ctrl キーをしながらフローにシンボルを追加すると、デフォルトのフロー方向を変更できます。

モデル要素からのアクティビティ シンボル

ドラッグ アンド ドロップで、情報を［モデル ビュー］からアクティビティ図にコピーすることもできます。たとえば、操作ノードをドラッグ アンド ドロップしてこの操作を参照するアクティビティ シンボルを作成できます。同じ操作を相互作用ノード、状態機械 ノード、およびユースケース ノードにも適用できます。

注記

アクティビティ ノードをアクティビティ シンボルにドラッグする前に参照から可視にするためには、ショートカット メニューを使用して既存のアクティビティ シンボルの［アクション］を選択する必要があります。

アクティビティ図のモデル要素

アクティビティ図には、以下の要素が使用されます。

- 開始ノード
- アクションノード
- オブジェクト ノード
- 分岐
- マージ
- フォーク
- ジョイン
- コネクタ
- イベント受信
- シグナル送信
- タイム イベント受信
- アクティビティ終了
- フロー終了
- アクティビティ区画
- ピン
- 関係

アクティビティ

アクティビティはユース ケース、操作その他振る舞いを持つエンティティの振る舞いを表すシグニチャです。アクティビティでは、振る舞いを小さい振る舞い単位、アクションノードに分類し、トークン フロー モデルをベースにしてこれらの単位の実行を制御します。アクティビティの実装は通常アクティビティ図によって記述されます。

シンボル

図 59: アクティビティ

構文

アクティビティ シンボルは、操作シンボルを基にしています。アクティビティの名前用の編集可能フィールドと、アクティビティのパラメータのための入力領域があります。

アクティビティに適用されたステレオタイプは、名前フィールドの上にあるテキスト フィールド（編集不可）に表示されます。

アクティビティ実装

アクティビティ実装は、アクティビティシグニチャの実装です。アクティビティ実装には、アクティビティ図と、アクティビティ エッジに接続している一連のアクティビティ ノードが含まれています。アクティビティ実装は、通常、アクティビティ生成時に暗黙的に生成されます。

トークンフロー

アクティビティ実装の実行セマンティクスは、トークン フロー モデルをベースにしています。トークンは、あるアクティビティ ノードから別のアクティビティ ノードに向け、接続されたアクティビティ エッジを通じて流れてゆきます。トークンには次の 2 つの種類があります。

- 制御トークン
- データ トークン（またはオブジェクト トークン）

アクティビティ エッジは両方の種類のトークンを転送できます。制御トークンがエッジを越えて転送されるときは制御フローを表し、データ トークンがエッジを越えて転送されるときはデータ フローを表します。制御フローは、オブジェクトノード以外の任意のアクティビティノードを末端に接続した、1 つのアクティビティエッジです。データフローは、エッジの末端のいずれかの側、または両側にオブジェクトノードを接続した、1 つのアクティビティエッジです。

コントロールトークンは、モデル化されたシステムの論理制御の状況を構成します。一方、データトークンは、モデル化されたシステム内を流れてゆくデータ単位の状況を現すために必要です。

1 つのアクティビティエッジとは、アクションノード、制御ノード、オブジェクトノード、ピン、コネクタと連結した、方向付きのエッジです。エッジの方向は、フローの方向を現しています。アクティビティエッジのセマンティクスは、そのターゲットノードとソースノードに依存します。

アクティビティが呼び出されると、そのアクティビティに含まれる各開始ノードに制御トークンが置かれて、アクティビティ実装の実行が開始されます。次に、これらのトークンは発信アクティビティ エッジを横切って下流方向に流れ、これらのエッジがつながっているアクティビティ ノードの着信アクティビティ エッジ側に集まります。アクティビティ ノードは、その**入力条件**が満たされるとすぐに実行を開始できます。アクティビティ ノードの種類によって入力条件が異なります。ただし、標準的な条件として、実行を開始するためには、各着信アクティビティ エッジに使用できるトークンが存在している必要があります。アクティビティ ノードがその実行を完了すると、（ある種の）トークンをすべての発信アクティビティ エッジ エンドに送信します。最終的に、これらのトークンはほかのアクティビティ ノードに届き、その手順が繰り返されます。

注記

アクティビティの実装は、トークンが流入している間は継続します。アクティビティ実装のどのアクティビティ ノードも入力条件を満たしていない場合、トークンは流れず、アクティビティ実装は実行モードのままです。つまり、制御はアクティビティの

呼び出し側には戻りません。特殊なアクティビティ ノードであるアクティビティ終了ノードが実行されたときのみ、全体のアクティビティ実装が実行を終了し、制御がアクティビティの呼び出し側に戻ります。

開始ノード

開始ノードは、アクティビティ実装の制御フローの開始点を指定します。アクティビティが呼び出され、実装の実行が開始すると、その実装の各開始ノードは制御トークンを受け取ります。

アクティビティ実装は開始ノードをいくつでも持つことができます。つまり、複数の制御フローを開始できます。また、開始ノードを持つことは必須ではありません。フローはピン、イベント受信、タイム イベント受信から開始することもできます。

開始ノードは着信アクティビティ エッジを持たない場合もあるので、入力条件はありません。開始ノードは、制御トークンを受け取るとすぐに実行を開始し、このトークンを発信エッジに渡します。

シンボル

図 60: 開始ノード

アクションノード

アクション ノードは、アクティビティ内の実行可能な機能です。アクション ノードの振る舞いは、アクティビティ、操作、または状態機械を使用するなど、多くの方法で指定できます。また、振る舞いをアクション モードに関連付けないようにすることもできます。これは、開発の初期の段階など、振る舞いの詳細が分からないときに役に立ちます。

アクション ノードに振る舞いがある場合は、そのアクション ノード内でインラインに定義できるか、またはそのアクション ノードから参照できます。インラインで定義された振る舞いは、合成的な階層のアクティビティ実装を指定する場合に適しています。合成状態と比較してください。参照される振る舞いは、モデル内の複数のアクション ノードで同じ振る舞いを再利用する場合に適しています。参照される振る舞いを使用する場合、通常ではアクティビティとなりますが、一般的には操作を参照することもできます。参照される振る舞いも実装を持つことができます。たとえば、参照されるアクティビティはアクティビティ実装を持つことができます。

トークンがすべての着信アクティビティ エッジで利用可能なとき、アクション ノードの入力条件が満たされます。その結果、トークンは消費されて実行が開始します。実行が終了すると、制御トークンがすべての発信エッジに与えられます。

実行時のデッドロックの回避

アクションノードへの入力条件が満足されない限り、そのアクションノードは実行できません。実行時のデッドロックを回避するには、アクティビティ実装でのトークンフローのセマンティクスを理解することが非常に重要です。一般的な誤解を解くための例として、217 ページの図 61 に示したアクティビティ実装を考えます。

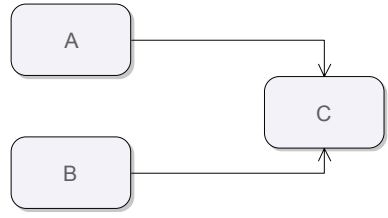

図 61: アクションノード間の制御フロー

この例には、3 つのアクションノード、A、B、C があり、2 つの制御フロー、A から C、B から C があります。ここで、C は両方のエッジにトークンがある場合のみ実行されます。A から C へのエッジにのみトークンがある場合は、C ノードは B から C へのエッジでトークンを待ちます。ノードはエッジ上に集められることを理解してください。

最低エッジの一方のみにトークンがあれば C ノードを実行できるようにしたい場合は、下図のようにノードの間に マージ ノードを挿入します。

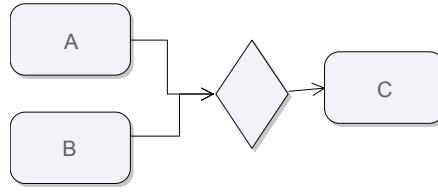

図 62: マージノード

ピン

アクション ノードは、振る舞いを持つ場合、その振る舞いへのパラメータを表すピンを持つことができます。トークンが着信アクティビティ エッジを通って直接アクション ノードに到達するか、接続されているピンを通って間接的に到達するかは重要です。直接到達する場合は、アクション ノードはその入力条件が満たされたときに実行されます。間接的に到達する場合は、アクション ノード自体は実行されません。代わりに、トークンが「ストリーミング」の形で振る舞いの実装に流入して、その結果、振る舞いの実装の実行は開始ノード上の制御トークンではなく、ピン上のデータ トーク

ンを使って開始されます。これらの 2 つの仕組みを組み合わせることが、制御トークンをアクション ノードに流入させ、そしてデータ トークンをそのピンに流入させることによって可能になります。振る舞いがその実行のためのデータを必要とするとき、これはよく用いられる設計の方法です。したがって、振る舞いは入力ピン上で入力データを取得し、実行が開始するときに制御のための制御トークンを取得します。アクティビティ最終ノードを実行して実行を終了する前に、通常、出力ピンにデータ トークンとして配置される出力データが与えられます。

ピンの詳細については、ピンを参照してください。

シンボル

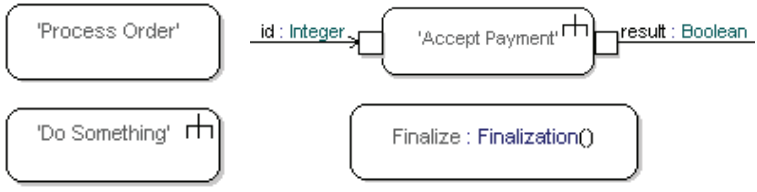

図 63: アクションノード－ピンあり／なし、振る舞いあり／なし（左：インライン、右：参照）

ショートカット メニュー選択項目として、[アクション] があります。チェックマークを付けると、テキストフィールドがアクション コードに追加されます。デフォルトでは、アクションテキストフィールドは表示しません。

ショートカット メニュー選択項目として、[区画参照] があります。このコマンドは、シンボル名フィールドの上に区画参照のテキスト フィールドを表示します。デフォルトでは、[区画参照] テキスト フィールドは表示されません。

ショートカット メニュー選択項目として、[すべてのパラメータの表示] があります。[すべてのパラメータの表示] コマンドで、現在、選択しているモデルに対してすべてのピン／パラメータ シンボルを表示できます。

構文

アクション ノード シンボルには非形式名を含むことができます。アクション ノードが振る舞いを参照する場合、振る舞いのシグニチャはコロンの後に表示されます。アクション ノードがインラインの振る舞いを含む場合は、シンボルの右上に「レーキ」のマークが表示されます。

アクション ノードを明示的に含むアクティビティ区画を、名前フィールドの上の別のテキスト フィールドに指定できます。この構文は、丸かっこで囲んだアクティビティ区画への参照をカンマで区切ったリストです。

アクション ノードに適用されたステレオタイプは、アクティビティ区画参照フィールドの上にあるテキスト フィールド（編集不可）に表示されます。

オブジェクト ノード

オブジェクト ノードは、フローに関与するクラスなどの分類子のインスタンスを表します。インスタンスとその値は、アクティビティで使用できます。

オブジェクト ノードの実行前に各着信アクティビティ エッジにトークンが存在しているとき、オブジェクト ノードの入力条件が満たされます。オブジェクト ノードの実行は、単純に、データ トークンを各発信エッジに配置することを意味します。データ トークンのタイプはオブジェクト ノードのタイプであり、すなわち分類子です。

オブジェクト ノードは出力データの取得方法は指定しません。取得方法を指定するには、出力ピンを持つアクション ノード ノードを使用できます。このアクション ノードの振る舞いはデータの計算方法を指定します。

シンボル

図 64: オブジェクト ノード

構文

オブジェクト ノード シンボルは、それが表す分類子の名前を含む 1 つのテキスト ラベルを持ちます。オブジェクト ノードに非形式名を付けることもできます。構文は <name> : <type> になります。

オブジェクト ノードに適用されたステレオタイプは、名前フィールドの上にあるテキスト フィールド（編集不可）に表示されます。

分岐

分岐ノードは、ガード条件に基づいて複数の外向きフローから 1 つを選択するためにフローで使用する制御ノードです。分岐ノードには、それぞれガードの付いた着信エッジが 1 つと発信エッジが複数あります。

トークンが分岐ノードの着信エッジに到達すると、発信エッジのガードが評価されます。「else」ガードが最後に評価されること以外、ガードを評価する順序は UML には定義されていません。従って、相互に排他的なガード条件を指定することを推奨します。最大で 1 つのガードは「else」ガードとなる可能性があります。このガード条件は、他のガード条件が満たされない場合でも満たされます。

入力トークンは、ガード条件を満たした最初のエッジに置かれます。そのようなエッジがない場合、トークンは分岐ノードによって破壊されます。通常、これは例外的な状況であり、「else」ガードをエッジの 1 つに設けて、この状況を避けることが望ましいです。

注記

アクティビティ シミュレータの活動実行セマンティクスでは、現在インフォーマル分岐と分岐回答のみをサポートします。このような分岐ノードを実行すると、どの発信エッジを選択するかをモデル ベリファイヤ（Model Verifier）が対話によって指示します。これによって、正確なガード条件が分かる前にアクティビティをシミュレーションできるので、これは開発の初期段階で有用な機能です。

シンボル

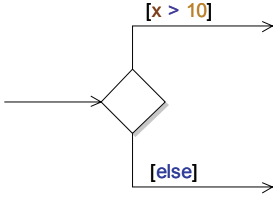

図 65: 分岐ノード

正式な定義では、ガード条件の値は論理表現になります。アクティビティ実装に可視変数、たとえばローカル変数などがある場合、それをガード条件に使用できます。キーワード `else` をガードに使用すると、他のガードの値がいずれも `true` にならない場合はこのエッジが選択されるよう指定されます。

複数の外向き分岐フローを 1 つのフローにマージし直すには、マージ ノードを使用します。

注記

分岐 ノードとマージ ノードには、アクティビティ図エディタのシンボル パレットの同じシンボルを使用します。

マージ

マージ ノードは、複数のフローを 1 つにまとめるのに使用する制御ノードです。トークンが着信エッジの 1 つに到達すると、そのトークンは発信エッジに中継されます。ジョインと異なり、内向きフローの同期化ではありません。

シンボル

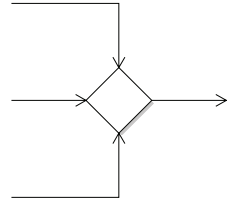

図 66: マージ ノード

注記

マージ ノードと分岐 ノードには、アクティビティ図エディタのシンボル パレット内の同じシンボルを使用します。

フォーク

フォーク ノードは、1 つのフローを複数の並行フローに分割する制御ノードです。トークンが入力エッジに到達すると、そのトークンのコピーが作成され、コピーが各発信エッジに配置されます。これにより、フォーク ノードはアクティビティ モデルに並列処理を導入するための手段となります。

複数の並行フローを 1 つのフローに結合し直すには、ジョインノードを使用します。

シンボル

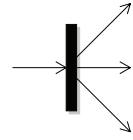

図 67: フォーク ノード

注記

フォーク ノードとジョイン ノードには、アクティビティ図エディタのシンボル パレット内の同じシンボルを使用します。

ジョイン

ジョイン ノードは、複数の並行フローを 1 つのフローにジョイン、つまり同期化し直す制御ノードです。

すべての着信エッジでトークンが利用可能なとき、ジョイン ノードの入力条件が満たされます。この条件が満たされると、次の規則に従ってトークンが出力エッジに置かれます。

- すべての入力トークンが制御トークンである場合、1 つの制御トークンが出力エッジに置かれます。
- 入力トークンのいくつかがデータ トークンである場合、これらのデータ トークンを除くすべてのトークンが出力エッジに置かれます。

注記

アクティビティ シミュレータ内でのアクティビティ実行セマンティクスの現在の実装は、この規則には従っていません。代わりに、ジョインに最後に到着するトークンによって、発信エッジに配置されるトークンの種類が決まります。

1 つのフローを複数の並行フローに分岐するには、フォーク ノードを使用します。

シンボル

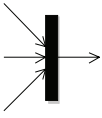

図 68: ジョイン

注記

ジョイン ノードとフォーク ノードには、アクティビティ図エディタのシンボル パレット内の同じシンボルを使用します。

コネクタ

コネクタ ノードは、複雑なフローの表記の図形的な省略表現です。アクティビティ エッジをコネクタ ノードで終了し、同じ名前の別のコネクタ ノードで継続させることができます。コネクタ ノードを使用して、アクティビティ実装仕様を複数のアクティビティ図に分割できます。

コネクタ ノードには複数の着信エッジを含むことができますが、含むことができる出力エッジは多くても 1 つです。セマンティック上、コネクタ ノードはマージ ノードと同じです。コネクタ ノードの着信エッジに到着するトークンは、その出力エッジに中継されます。コネクタ ノードに出力エッジを含まない場合、セマンティック上、コネクタ ノードはフロー終了 ノードと同じです。

シンボル

図 69: コネクタ ノード

構文

コネクタ ノード シンボルには、コネクタ ノードの名前が入るテキスト ラベルがあります。

イベント受信

イベント受信ノードは、特定のイベント、通常はシグナルを待っていることを示すために使用します。特定のイベントが受信されると、制御トークンをすべての発信エッジに配置することによって、フローが継続します。

セマンティック上、イベント受信ノードは、受信対象のイベントを待ち受ける振る舞いを持つアクション ノードノードと同じです。

このイベントで渡されるデータは、イベント受信ノードからの出力ピンを使用して、後でフロー内で使用できます。受信イベント ノードには入力ピンを含まないことも可能です。

イベント受信アクションは、状態機械のシグナル受信（入力）に似ています。

シンボル

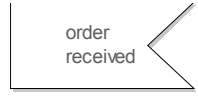

図 70: イベント受信ノード

シグナル送信

シグナル送信ノードは、シグナルのインスタンスを生成して送信するために使用します。これは状態機械のシグナル送信アクション（出力）に似ています。

セマンティック上、シグナル送信ノードは、シグナルを送信する振る舞いを含むアクション ノードノードと同じです。

シグナル送信ノードには、送信対象の信号のフォーマル パラメータの実際の引数を与える入力ピンを含むことができます。シグナル送信ノードには出力ピンを含むことができません。

シンボル

図 71: シグナル送信シンボル

タイム イベント受信

タイム イベント受信はイベント受信ノードの特殊なバージョンです。タイム イベント受信は、特定のタイム イベント、通常はタイマーのタイムアウトまたは絶対時間値を待っていることを示すのに使用します。特定のタイム イベントが受信されると、制御トークンをすべての発信エッジに配置することによって、フローが継続します。

イベント受信ノードとは違って、タイム イベント受信ノードにはピンを含むことはできません。パラメータを持つタイマーを待つためには、代わりにイベント受信ノードを使用してください。

シンボル

図 72: タイム イベント受信

アクティビティ終了

アクティビティ終了ノードは、アクティビティの終わりを示します。トークンがアクティビティ終了ノードに到達すると、アクティビティのすべてのフローが終了し、アクティビティの実行が完了します。制御はアクティビティの呼び出し側に戻ります。

アクティビティ終了ノードには任意の数の入力エッジを含むことができますが、出力エッジを含むことはできません。

アクティビティ内の 1 つのフローを終了するには、フロー終了ノードを使用します。

シンボル

図 73: アクティビティ終了

フロー終了

フロー終了は、アクティビティ内の 1 つのフローの終わりを示します。アクティビティ全体ではなく、特定のフローのみが終了します。アクティビティ内に進行中の他のフローが存在する場合があります（フォークと比較してください）。

フロー終了ノードが受け取ったトークンはそのノード内で消費されます。1 つのフロー終了ノードは任意の数の入力エッジを持つことができますが、出力エッジを持つことはできません。

アクティビティ全体を終了するには、アクティビティ終了ノードを使用します。

シンボル

図 74: フロー終了

アクティビティ区画

アクティビティ区画は、スイムレーンとも呼ばれ、関連アクションノードを互いにグループにまとめる仕組みです。アクティビティ図を複数のセクションに分割でき、これで、特定のアクティビティを実行するセクションや、セクション間のデータ フローを確認しやすくなります。

たとえば、ビジネス モデリングでは、会社の様々な部門をそれぞれ区画で表すことができます。他の例としては、リアルタイム オペレーティング システムのスレッドを区画で表すことができます。これで、ダイアグラムには、システムのアクションがスレッド間でどのように分散されているかが示されます。

アクティビティ区画には、通常、クラスと呼ぶタイプを含むことができます。このことは、アクティビティ区画のアクションを実行するインスタンスはこのタイプのインスタンスでなければならないという制約を表しています。このアクティビティ区画は、後でアクションを実行する特定のインスタンスを指定することによって、実行するアクションをさらに制約できます。また、このアクティビティ区画は、アクションを実行するインスタンスを含む属性を指定できます。

シンボル

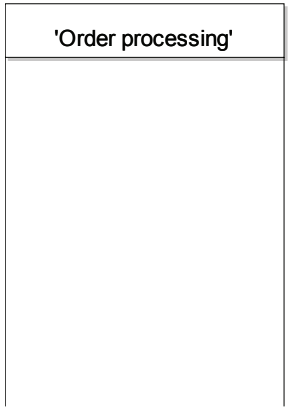

図 75: アクティビティ区画

アクティビティ区画のタイプ、インスタンスまたは属性に関する制約は、名前ラベルの真下にあるラベルに指定します。構文は、ライフライン（生存線）に使われる構文と同じです。

アクティビティ区画 シンボルに図形として含まれるアクションノードノード シンボルは、そのアクティビティ区画に所属するアクションを表します。1 つのアクション ノードを複数のアクティビティ区画に所属させることができます。これは、アクティビティ区画 シンボルを回転するときに起こり得ます。この結果、2 つのアクティビティ区画 シンボルの交差部分に同じアクション ノード シンボルが含まれます。ただし、アクティビティ図は 2 次元なので、この方法で、複数のアクティビティ区画への関与を実現することはできません。アクション ノードが複数のアクティビティ区画に所属するように指定するため、含まれる区画の明示的なリストをアクション ノードについて指定できます。アクション ノードにアクティビティ区画参照の明示的なリストを含む場合、このリストは図形としての位置から推定できる暗黙の参照に優先します。

例 32: 暗黙的または明示的アクティビティ区画参照

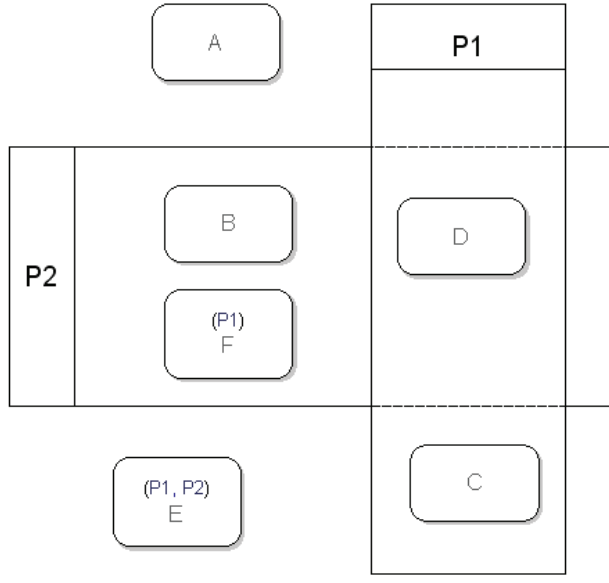

図 76: アクティビティ区画を参照するアクション ノード

上記のアクション ノードは、暗黙のアクティビティ区画参照（アクション ノード シンボルの図形的な位置から推定）と明示的な区画参照の両方を使用します。

A はどの区画にも所属しません。

B は区画 P2 に所属します（暗黙の参照）。

C は区画 P1 に所属します（暗黙の参照）。

D は区画 P1 と区画 P2 に所属します（暗黙の参照）。

E は区画 P1 と区画 P2 に所属します（明示的な参照）。

F は区画 P1 に所属します（明示的な参照）。

ディメンション指定シンボルとしての区画 シンボル

区画 シンボルに複数行がある場合、最上部にある区画 シンボルはディメンション指定シンボルとして使用される可能性があります。その区画 シンボルにディメンションを選択できるショートカット メニューがあります。ディメンションを選択した場合、区画 シンボルのメインのラベルがイタリック体になります。

水平ディメンションと垂直ディメンションの両方を同時に使用できます。

ピン

ピンは、第 i 章「UML 言語ガイド」の 216 ページ、「アクション ノード」ノードの振る舞いのパラメータを表し、振る舞いとの間のデータの受け渡しに使用します。ピンは、アクションとの間の入出力用のオブジェクト ノードと見なすことができます。

着信エッジを持つピンはデータを振る舞いに入力するので、入力ピンと呼ばれます。発信エッジを持つピンはデータを振る舞いから出力するので、出力ピンと呼ばれます。ピンが表すパラメータの方向は、エッジがピンに接続される方法と一致しなければなりません。たとえば、入力ピンは着信エッジだけを含み、対応するパラメータの方向は「in」である必要があります。

ピンの実行のセマンティクスはオブジェクト ノードの場合と同じです。従って、実行により各発信エッジにデータ トークンが配置され、これらのデータ トークンのタイプはピンが表すパラメータのタイプです。

ピンはストリーミングにも非ストリーミングにもなることができます。ストリーミングの場合、アクション ノードノードの振る舞いが実行しているときでも、ピンは出力データ トークンの作成を実行できます。実際に、ストリーミング入力ピン上のトークンの存在とアクション ノードノードの振る舞いが呼び出されるときの条件との間に関係はありません。ただし、非ストリーミングの場合、トークンがすべての入力ピンで利用できるようになるまで振る舞いは実行されません。

注記

アクティビティ シミュレータ内でのアクティビティ実行セマンティクスの現在の実装は、ストリーミング ピンをサポートするだけです。ただし、アクション ノードノードの振る舞いのアクティビティ実装で、ストリーミング ピンとジョイン ノードを結合することにより、非ストリーミング ピンをエミュレーションできます。次に、ジョイン ノードには 2 つの着信エッジがあります。1 つはデータ トークンが到着するピンからの着信エッジで、もう 1 つは、振る舞いを実行するときに制御トークンが到着する開始ノードからの着信エッジです。

シンボル

図 77: ピン シンボル

構文

ピン テキストの構文はパラメータと同じで、name :Type となります。

関係

アクティビティ エッジ

アクティビティ エッジは、アクティビティ実装内のノードを接続するために使用します。アクティビティ エッジは、接続された 2 つのノード間の制御トークンとデータ トークンのフローを可能にします。

アクティビティ エッジは必ず方向を持っています。すなわち、トークンはアクティビティ エッジ上を 1 方向にのみ流れることができます。エッジの方向は流れの方向を現します。アクティビティ エッジは両方の種類のトークンを転送できます。制御トークンがエッジを越えて転送されるときは制御フローを表し、データ トークンがエッジを越えて転送されるときはデータ フローを表します。

アクティビティ エッジには、アクティビティ エッジが表すフローを説明する非形式名を含むことができます。

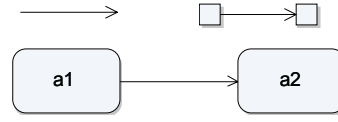

図 78: アクティビティ エッジ

振る舞いモデリング

実行可能なモデルを得るには、操作とアクティブ クラスの詳細な振る舞いを決定する必要があります。仕様決定は振る舞いモデリングで行います。このアクティビティは通常、設計フェーズの最後に行われます。

振る舞いの仕様にステートが含まれる（状態機械実装）ことも、ステートレス（操作本体）であることもあります。いずれの場合も、振る舞いの記述には、以下の 2 つの方法があります。

- 状態機械図の状態機械として記述する

ステートを含む実装の場合はグラフィック形式（状態機械図）のほうが好ましい場合が多く、操作の単純な実装であれば、操作本体を構成するアクションのテキスト記述で十分なことがあります。

状態機械図

状態機械図は、状態機械を可視化したものです。サポートされている状態機械図の作成スタイルは 2 つあります。スタイルについては以下で説明し、例を示します。2 つのスタイルを組み合わせることもできます。

状態（ステート）指向ビュー

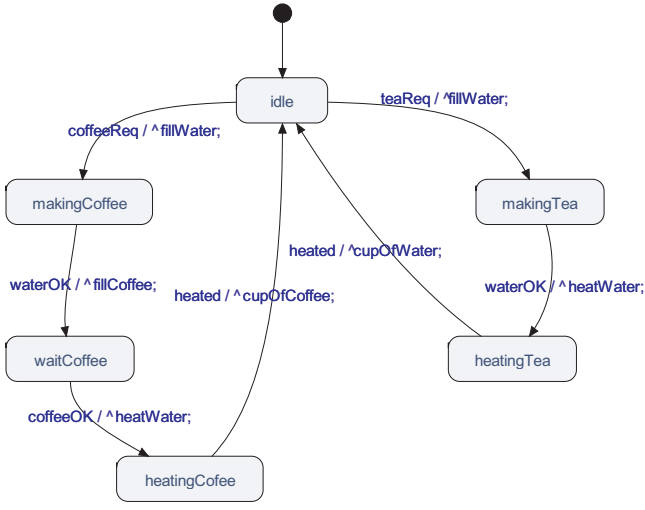

図 79: 状態機械の状態（ステート）指向ビュー

状態機械の状態（ステート）指向ビューにより、複雑な状態機械の概観ができますが、特定の遷移の制御フローや通信の側面に焦点を当てている場合はあまり実用的ではありません。そのため、状態機械を、遷移中に実行できるさまざまなアクションに明示的なシンボルを使用して、遷移指向で記述することも可能です。

遷移指向ビュー

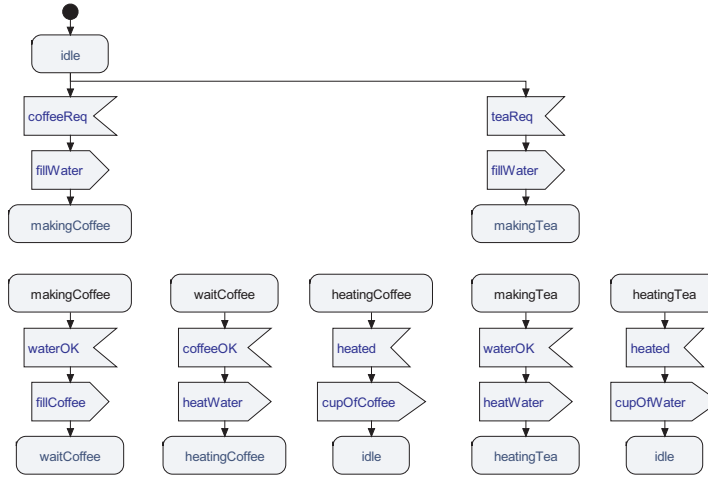

図 80: 状態機械の遷移指向ビュー

状態機械図の作成

状態機械図はクラスと操作（ユースケース含む）に含まれます。

1. ［モデル ビュー］から状態機械を作成する場所となるエンティティを選択します。
2. ショートカット メニューから［新規］を選択し、［状態機械図］を選択します。

状態機械

UML 状態機械は、データとシグナルのハンドリングで拡張された有限状態機械です。状態機械の基本要素は、ステートと遷移です。状態機械 パラダイムに基づくモデルでは、実行は開始点としての何らかのステートと、遷移を実行する起動イベントによって行われます。遷移では、アクションを実行できます。遷移の終わりに、新規ステートに入ります。状態機械は、遷移を開始する新たな起動イベントが発生するまで、このステートでアイドルの状態です。遷移を終了する他の方法として、状態機械（アクティブ クラス）全体を停止します。

ヒント

状態機械は、［モデル ビュー］でクラスを右クリックしてショートカット メニューで［新規］->［状態機械図］を選択するか、［プレゼンテーションの作成］ダイアログを開いて作成します。

シンボル

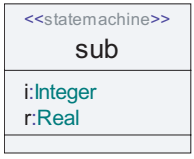

図 81: 状態機械

構文

シンボルには、以下に示すように、編集可能なテキスト フィールドが 2 つあります。

- クラス ヘッダー
- パラメータ

（操作フィールドは空白です。）

パラメータフィールドには、状態機械の仮パラメータが入ります。パラメータは以下の目的で使用されます。

- 生成時のアクティブ クラス インスタンスへの値の受け渡し
- 合成状態へ入った際の値の受け渡し

ステート

ステートは、包含するオブジェクトが、別のステートへの遷移のトリガとなるイベントを待っている状態機械の状況を表します。状況には、静的状態があることがあります（ステートにサブステートがない場合）。この場合、状態機械は、その状態にある間は非アクティブです。状況は、ステートのサブステートに状態機械の振る舞いが隠れているという意味で、動的であることも考えられます。

シンボル

図 82: ステート

ステート シンボルは 1 つまたは複数のステートを参照し、このステート（または一連のステート）からの遷移のターゲットや、ステートへの遷移のソースとして機能します。

構文

- 単純なステート

  `State1`
- リストのあるステート

  `St1,st2`
- 含まれていないステートのリストを含む、アスタリスク ステートのあるステート

  `*(st1,st2)`

アスタリスク ステートは、* シンボルに続くリストで挙げられたステートを除き、現在の状態機械で定義されたすべてのステートを参照するショートカットです。

状態機械は階層的であるため、ステートにはサブ状態機械を持たせることができます。これは、233 ページの図 83 に示すように、ステートの名前に続くコロンの後に状態機械の名前を指定することによって、示すことができます。

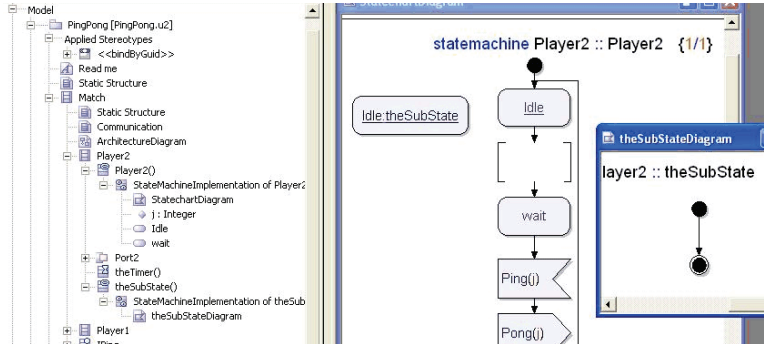

図 83: サブステートの参照

`<state>:<state machine>` 構文を使用できるのは、着信ラインのないステート シンボルの場合のみです（ステート シンボルが「nextstate」ではない）。ラベル `s:myStatemachine` 付きのステート シンボルに着信ラインがある場合は、構文エラーになります。

注記

ステート シンボルには、そのステート シンボルをターゲットとする遷移がある場合は、ステートのリストやアスタリスク ステート定義を含めることができません。ステート リストとアスタリスク ステートでは、遷移のソースのみを指定でき、遷移のターゲットは指定できません。

ステートにサブステート 状態機械があり、この状態機械にエントリ ポイントがある場合、そのエントリ ポイントはステート シンボルで表すことができます。これが可能なのは、ステート シンボルをターゲット ステートとする遷移が 1 つのみの場合だけです。

例 33: via 句を含むステート

サブステート 状態機械のエントリ ポイントを決める via 句を含むステート St1

```
St1 via entry1
```

状態機械がサブステートを持つ場合は、フローの分岐を表すマークとしてシンボルの右上に「レーキ」のマークが表示されます。234 ページの図 84 を参照してください。

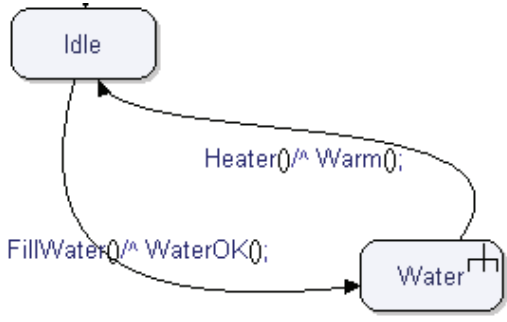

図 84: サブステートを持つステート

ステート シンボルはステートの定義とステートの参照（遷移のターゲット ステート）の両方に使用できるため、シンボルを利用して、遷移の終点と、新規遷移の開始点の両方にし、遷移をチェーンにできます。これは、状態機械の状態（ステート）指向レイアウトを使用している場合は便利です。ただし、遷移指向レイアウトを使用している場合は、読みやすくためにこれを避け、遷移を常に分離するほうがよいでしょう。上述の例を参照してください。

ステートが多数の遷移のソースになる場合、読みやすくするために、これらを複数のダイアグラムで指定してもかまいません。ステート シンボルはステートの部分定義です。

同じ遷移が複数のステートで有効な場合は、ステート シンボルから複数のステートを参照できます。

参照

合成状態

遷移

遷移は、状態機械がアクティブ ステートを変更する際に実行されるアクションのシーケンスです。

遷移に使用する構文は、状態（ステート）指向構文を使用するか遷移指向構文を使用するかにより、2 つのカテゴリに分類されます。状態（ステート）指向遷移構文については 251 ページの「シンプル遷移」で説明しています。遷移指向構文は、遷移開始のための一連のトリガ シンボル、および遷移の詳細を記述する一連のアクション シンボルで記述します。

複数のトリガ シンボルが、遷移を開始させるイベントに対応しています。これに基づき、さまざまな種類の遷移があります。

- トリガ付き遷移
- ガード付き遷移
- ラベル付き遷移
- 開始遷移

トリガ付き遷移には、遷移と関連付けられたトリガがあります。通常、このトリガは特定のシグナルによって定義されますが、タイマーや操作などによっても定義できます。トリガ付き遷移の詳細については、237 ページの「シグナル受信（入力）」セクションを参照してください。

ガード付き遷移の特徴は、特定のイベントにトリガ*されない*ということです。イベントの代わりに、true または false の条件（ガード）によってトリガされます。

ラベル付き遷移は、ステートごとの振る舞いの記述という意味では本当の遷移ではありません。ラベル付き遷移は、ダイアグラムで異なる 2 ページに記述できるよう、遷移を 2 つ（またはそれ以上）のパートに分解するのに使用します。ジャンクションも、フローを分割するのに使用する、ラベル付き遷移の関連構成要素です。

開始遷移（開始）は、状態機械生成時に直接実行される遷移です。

遷移は常に、停止、リターン、別の遷移への制御の移転により、状態機械がステートに入ると終了します。

ガード付き遷移

ガード付き遷移にはトリガがある場合とない場合があります。

ガード付き遷移にトリガがある場合、トリガ イベントが発生した後に式の値が求められます。式の値が true の場合、遷移が起こります。式の値が false の場合は、状態機械はステートにとどまり、トリガ イベントの原因となったシグナルがシグナル キューに置かれます。

参照

保存

履歴の次のステート

履歴の次のステートは、すぐ前のステートに戻るために、遷移の終わりに使用します。

シンボルは、単純な遷移とフロー ライン（詳細）遷移の両方を終了するために使用できます。

浅い履歴

デフォルトでは、履歴の次のステートは、**浅い**ものです。これは、History の付いた Nextstate が遷移の終わりで解釈されると、次のステートは、現在の遷移がアクティブ化されたステートになるということです。

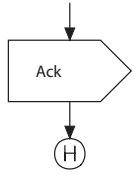

図 85: 浅い履歴の次のステート

履歴の次のステートは、シンボルで名前の代わりにハイフンを使用することにより、通常の Nextstate でも表現できます。

```
nextstate -;
```

詳細な履歴

履歴の次のステートを**詳細な**ものにすることもできます。浅い履歴と同様に、次のステートは、現在の遷移がアクティブ化されたステートになります。これは、入ったステートのサブステートのすべてのレベルまで繰り返し適用されます。

ヒント

履歴の次のステートは、選択し、ショートカット メニューから［詳細な履歴］コマンドを選択することによって深くできます。

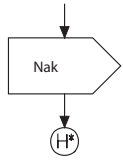

図 86: 詳細な履歴の次のステート

詳細な履歴の次のステートは、次の構文を使用して、通常の Nextstate でも表現できます。

```
nextstate ^-;
```

例

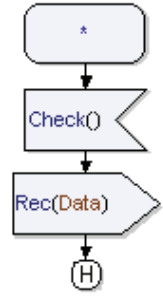

図 87: アスタリスク ステート遷移のある浅い履歴の次のステート

上述の例では、遷移は、遷移がトリガされた際にアクティブであったステートになります。

シグナル受信（入力）

シグナル受信シンボルは、特定の遷移をトリガするシグナルを定義します。

遷移は、ガード式でガードすることもできます。ガード式はシンボルに示されます。

シンボル

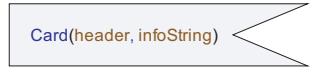

図 88: シグナル受信

シグナル受信シンボルはシグナルを受信し、常にステート シンボルが前になければなりません。両方合わせて遷移を定義します。

ヒント

ショートカット メニューから、シンボルを水平に反転させることもできます。
シグナル受信シンボルを削除すると、続くサブツリーも削除されます。

同じ遷移振る舞いを 1 つのステートで複数のトリガ用に呼び出す場合、シグナル受信シンボルに、識別子のリストを持たせることができます。この仕組みでは、各シグナルのパラメータのハンドリングはできず、すべてのシグナルが、1 つの Nextstate で終わる同じ遷移のトリガとなります。

シグナルを受信すると、パラメータは通常、ローカル変数に格納されます。また、パラメータを無視することもできます。

オプションのガード式はトリガの後に定義し、角かっこで囲みます。

シグナル キュー

状態機械は、状態機械に送られるシグナルを到着順に格納する**シグナル キュー**に関連付けられます。

各ステートの、考えられるすべてのトリガに遷移を指定する必要はありません。モデリングしようとしているアプリケーションやドメインの知識から、どのシグナルが到着するかを予測することが可能な場合がよくあります。シグナル キューで次に処理するシグナルが現在のステートで処理されない場合、シグナルは破棄されます。シグナルを一時的に保存することもできます。

構文

シグナル受信でトリガとして以下の種類を参照できます。

- シグナル
- タイマー
- 操作

例 34: 単純なシグナル受信

```
s1( i )
```

例 35: 複数のトリガのあるシグナル受信

```
s1(i), myTimer, s3
```

例 36: 仮想性のあるシグナル受信

```
redefined input s1( i )
```

例 37: アスタリスクシグナル受信

すべてのトリガが遷移を起動できるものと指定することもできます。これは、アスタリスクを使用してすべての可視トリガを指定することによって行います。

```
*
```

例 38: ガード付きシグナル受信

```
s1 [ x>10 ]
```

開始

開始シンボルは、状態機械の開始点または合成状態の開始点 1 つを定義します。したがって、開始シンボルにより、開始遷移が定義されます。

シンボル

図 89: 開始

構文

開始シンボルには、以下の目的で使用できるテキスト フィールドが 1 つあります。

- 合成状態のエントリ ポイントの参照

  ```
  Entry1
  ```

- 遷移の仮想性の定義

  ```
  virtual
  virtual Entry2
  ```

アクション

アクションは、通常、テキスト構文を使用してアクション シンボルで行います。使用できるアクションは以下のとおりです。

- ローカル変数の定義文
- 空文
- 複合文
- 代入
- アクション
  - Signal Sending（output）
  - New
  - Set
  - Reset
- 式文
- If 文
- 分岐文
- ターゲット コード文
- While 文
- For 文

- Delete 文
- Try 文
- 終了文
  - Return
  - Break
  - Continue
  - Stop
  - Nextstate
  - Goto（join）
  - Throw

これらの文の一部には、グラフィック構文、つまり専用シンボルもあります。stop、return、分岐、シグナル送信の各文には、遷移に対する重要な操作を強調できるようにするための個別シンボルがあります。当然、これらの文にテキスト構文を使用することもできます。最重要アクションについて説明します。

シグナル送信アクション（出力）

遷移でのシグナル送信アクションにより、シグナルを別の状態機械、環境、または同じ状態機械内に送ることができます。シグナルにパラメータがある場合は、パラメータ タイプに一致する文を指定する必要があります。シグナル送信時、パラメータを無視してもかまいません。

1 つのシグナル送信に複数のパラメータを指定することもできます。これで、連続した別個のシグナルの送信として処理されます。

シンボル

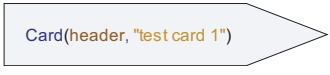

図 90: シグナル送信

シグナル送信シンボルは、遷移からシグナルを送ります。

ヒント

ショートカット メニューから、シンボルを水平に反転させることもできます。

シグナル アドレッシング

以下に示すように、シグナルを受信側にダイレクトしたり、ルーティングしたりする方法にはいくつかあります。

- アドレッシングを省略する
- シグナルを受信側に対するメソッド アプリケーションとしてダイレクトする
- ポートまたはインターフェイスを通じてシグナル送信する

これらのアドレッシング メカニズムそれぞれについて説明します。シグナルの直接アドレッシングは、受信側に対するメソッド アプリケーションでは、ピリオド（<receiver>.<signal>）を使用して表現されます。

シグナル送信

アドレスやパスは指定しません。シグナルは、考えられるパスのいずれか（ポート／コネクタ）で送信されます。

受信側が this の場合

コンテキストがアクティブ クラスの状態機械または操作の場合、**this** とは現在のアクティブ インスタンスの状態機械、つまり自分と同じことです。

コンテキストがパッシブ クラスの操作の場合は、代わりに **self** を使用して現在のインスタンスの状態機械を参照します。この場合、**this** はパッシブ クラスのインスタンスを指します。

ポートまたはインターフェイスを通じたシグナル送信

ポート識別子が指定されます。シグナルは、このポートを通じて送信されます。

インスタンスを 1 つだけ実現化する匿名ポートがクラスに定義されている場合は、識別子をインターフェイス名にすることもできます。この場合、その匿名ポートが参照されます。

受信側が属性の場合

変数または属性が宛先として指定されます。変数や属性のタイプはインターフェイス（これを通じてシグナル送信される）またはアクティブ クラス（または RTUtilities パッケージで定義された特殊タイプ Pid）でなければなりません。

属性により、暗黙的属性 **self**、**sender**、**parent**、**offspring** の 1 つを参照することもできます。

受信側が式の場合

式のタイプは、インターフェイスまたはアクティブ クラス（または RTUtilities パッケージで定義された特殊タイプ Pid）でなければなりません。これは受信側が属性の場合と似た状況です。違いは、フィールドや文字列抽出など、より複雑な式をかっこ内で指定できるという点です。

例

例 39: アドレッシング メカニズム

アドレスやパスの指定なし

```
SuspendInd
```

受信側が暗黙的属性

```
sender.Ack(id)
```

受信側が属性で、シグナルにパラメータあり

```
Bank.Card(carddata)
```

受信側が Pid 式（Pid 要素のあるインデックス付き配列）

```
(myList[10].addr).Sig1
```

インターフェイス（ポートを参照）

```
Ack(id) via myInterface
```

これらのすべてのアドレッシング メカニズムには以下の共通点があります。

- 状態機械の、生きているインスタンスが通信経路の終わりにない場合、シグナルは消失します。
- 宛先が、中止された状態機械 インスタンスを参照する場合、シグナルは消失します。
- 受信側状態機械が、シグナルが処理されないステートにある場合、シグナルは消失します。

分岐

分岐構成要素は、遷移で、式の値によって選択されたアクションを実行するために使用します。これはスイッチと似た仕組みです。分岐には**質問**パートが 1 つあり、ここに、分岐実行時に値が求められる動的な式が含まれます。また、分岐には複数の**回答**パートがあり、それぞれが範囲式（または値または定数を含む単純式）を持ち、複数の部分遷移となります。

シンボル

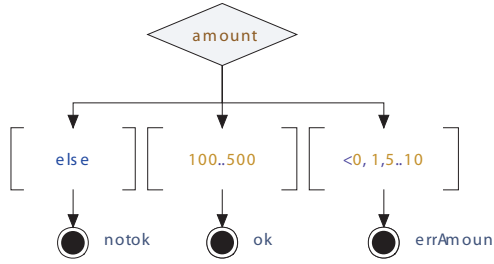

図 91: 分岐の使用

分岐シンボルは、遷移の振る舞いパートの選択パスを指定します。

- 式は以下のようになります。式を定義する必要があります . 各パスには、使用するパスの式に一致する回答でラベルを付けます。
- 分岐シンボルを削除すると、続くサブツリーも削除されます。

分岐回答

分岐回答シンボルは、遷移の振る舞いパートの選択パスのうち 1 つを指定し、分岐質問の回答となる範囲条件を含みます。

範囲条件は以下のいずれかで指定します。

- 特定の値（たとえば `10` や `true`）
- 一端が決まっている範囲（たとえば `>10`）
- 両端が決まっている範囲（たとえば `2..10`）
- 上述の選択のカンマ区切りリスト

インフォーマル分岐

モデルを早期にベリファイしやすくするため、インフォーマル分岐を指定できます。これらの分岐は、文字列と、文字列である回答のある式を持っています。

非決定性分岐

非決定性分岐を記述することもできます。これは、any（引用符なし）を使用し、分岐の回答を空白にしておくことによって行います。

構文

例 40: 分岐式テキストの例

```
v+4
```

例 41: 分岐選択肢テキストの例

単純な例

```
True
```

一端の決まった範囲

```
>10
```

両端の決まった範囲

```
0..3
```

複数の範囲

```
<-5, 0..2, >10
```

ガード

ガード シンボルは、以下の目的で使用できます。

- 一定の条件の値が true になった場合に遷移をトリガする
- 接続遷移、つまり終了ポイントによってサブステートから出る

シンボル

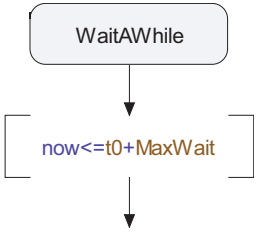

図 92: ガード付き遷移

条件に基づくガード付き遷移の場合、遷移は、条件を定義する式の値が true になると呼び出されます。この式は、単純な式で、副作用を発生させないものである必要があります。

遷移が、終了ポイントの参照によって定義される場合、遷移のソース ステートには、サブステート 状態機械がなければなりません。遷移は、このサブステート 状態機械が、指定された終了ポイントを通って終了するたびに実行されます。

構文

例 42: ガード付き遷移

```
[ x>10 ]
```

例 43: 接続遷移

名前のある終了ポイント a を通って合成状態が終了するとトリガされる遷移

```
[ a ]
```

例 44: 接続遷移

名前のない終了ポイントを通って合成状態が終了するとトリガされる遷移

```
[ ]
```

タイマー設定アクション

タイマー設定アクションにより、タイマー インスタンスが生成され、アクティブになります。アクティブなタイマー インスタンスに対して設定アクションを再度実行すると、初めのタイマー インスタンスが暗黙的にリセットされ、新規タイマー インスタンスが生成されます。

パラメータのあるタイマーの場合、個別のパラメータ値があれば、複数のタイマー インスタンスを同時にアクティブにできます。

構文

例 45: 絶対時間

```
set (MyTimer, aTime);
```

例 46: 相対時間

```
set (MyTimer, now+10);
```

例 47: デフォルト持続時間のあるタイマー

```
timer MyTimer () = 5;
...
set(MyTimer);
```

例 48: パラメータのあるタイマー

```
timer MyTimer (Integer id);
Integer i = 1;
...
set (MyTimer (i), now+5);
```

参照

タイマー アクティブ式

タイマー リセット アクション

タイマー リセット アクションにより、アクティブなタイマー インスタンスがあればリセットされます。

構文

例 49: 通常のタイマーのリセット

```
reset(MyTimer);
```

例 50: パラメータのあるタイマーのリセット

```
reset(MyTimer(i));
```

アクション（タスク）

アクション シンボルは、変数代入、for ループ、値を返すプロシージャのコールなど、遷移の振る舞いパートにテキスト コードを作成するために使用します。

シンボル

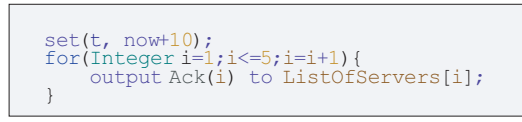

図 93: アクション シンボル

構文

例 51: 単純な例

```
Integer v1;
v1 = 4;
output s(v1);
```

代入

代入は、次に示す例の構文に従って行われます。代入の左側には変数識別子、インデックス付き変数の要素、構造体またはクラスの構造体フィールドを入れることができます。右側には、左側と同じタイプの式が入ります。

例 52: さまざまな代入

```
Integer i = 0;
myObject = new (theType);
person.age = person.age+1;
arrival[currentDate,person] = now;
```

代入は、それ自体を式として使用することもできます。代入が成功した場合、代入式で返された値が右側の式です。

例 53: 代入式

```
if ((a=10)==10) { output s; };
```

複合文

複合文には、中かっこ {} で囲まれた文が複数含まれます。また、名前空間も複合文によって定義されます。これで、複合文中でローカル変数を宣言できるようになります。

New

new 文は、アクティブ クラスとパッシブ クラス両方のインスタンスの生成に使用します。現在のクラスと同じクラスのインスタンスを生成するために、キーワード **this** を使用できます。この構成要素は、生成されたオブジェクトの参照を返します。

オブジェクトへの参照を使用して、作成されたインスタンスと交信することは常に可能です。new 文の実行結果を参照属性に割り当てることで、たとえばシグナルを作成されたインスタンスに直接送信したり、インスタンス上の操作を呼び出したりできます。

しかし、モデルに存在するポートやコネクタを使用してインスタンスと交信できるようにするには、アプリケーションに存在するアーキテクチャ（コネクタとポートの構成要素）に作成されたインスタンスを追加する必要があります。

保存

着信するシグナルを、一定の順序で処理する必要があることはよくあります。しかし、外界から着信するシグナルは、予期した順序で着信するとは限りません。他のシグナルの処理を待つ間、シグナルをシグナル キューに一時的に保存するため、セーブ シンボルを使用します。

各ステートに複数のシグナルを保存できますが、保存されたシグナルが次のステートで処理されない場合、破棄されることがあります。

シンボル

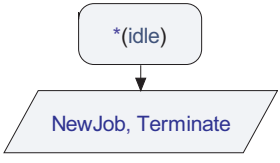

図 94 セーブ シンボルの使用

セーブ シンボルは、シグナルを処理しないステートで処理されそうな場合に、シグナルが破棄されないようにします。

- このシンボルの前には必ずステート シンボルが必要です。
- セーブ シンボルの後にはシンボルを挿入できません。

構文

例 54: 保存

単純な例

```
save s;
```

アスタリスク保存

```
save *;
```

停止

ストップ シンボルは、現在のインスタンスの実行を停止します。アクティブ クラスのインスタンスの削除は、クラスの状態機械内から、停止アクションを実行することによってのみ可能です。

シンボル

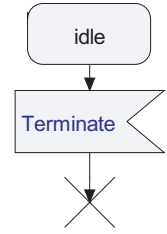

図 95: 停止

停止アクションは、次のように処理されます。

1. インスタンスがパートのない単純な状態機械の場合、状態機械は直ちに停止します。
2. インスタンスにパートがある場合、そのインスタンスに加え、各パート インスタンスが上述の 1 に従って処理されます。

リターン

リターン シンボルは、操作やサブステートの実行を終了し、呼び出しコンテキストに制御を移します。

シンボル

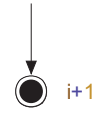

図 96: 操作のリターン

構文

例 55: リターンの単純な例

```
4+r
```

例 56: 合成状態の終了ポイント名のあるリターン

```
exP2
```

操作にリターン タイプがない場合や、合成状態終了がデフォルト終了ポイントで行われる場合、テキスト フィールドを空白にします。

ジャンクション

通常、複雑な状態機械を複数のダイアグラムに分割するにはステートと Nextstate で十分です。しかし、遷移が非常に長い場合、遷移の記述を複数のパートに分割する必要があることがあります。これは、ジャンクション シンボルによって行うことができます。ジャンクション シンボルは、ラベルと jump 文の両方として使用します。他に、複雑なフローで交差フロー ラインを避けるためにもジャンクションを使用できます。

シンボル

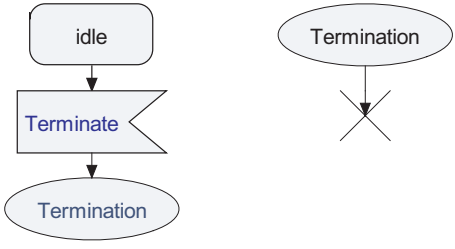

図 97: ラベルまたは goto としてジャンクションを使用

ジャンクション シンボルは、ラベルとジョイン シンボルに相当しますが、フロー ラインをマージする必要がある場合にも必ず使用します。

- ジャンクション シンボルには、複数の内向きフロー ラインを持たせることができます。

構文

シンボルには、テキスト フィールドが 1 つあります。

フロー

フロー ラインは、遷移の 2 つのシンボルを接続します。

- ドローイング エリアでシンボルが選択されており、<Shift> キーを押しながら別のシンボルをツールバーから追加した場合、シンボルの間に自動的にフロー ラインが生成されます。
- ライン ハンドルから次のシンボルにラインを接続することによってもラインを生成できます。
- シンボルを削除すると、シンボルに接続されたラインも削除されます。

シンプル遷移

シンプル遷移ラインは、ステート指向スタイルを使用している場合に遷移を定義するために使用します。

- シンプル遷移ラインは、ステート シンボルからのみ引くことができます。
- ライン ハンドルから次のシンボルにラインを接続するとラインを生成できます。
- シンボルを削除すると、シンボルに接続されたラインも削除されます。

構文

シンプル遷移ラインに関連付けられたテキスト フィールドが 1 つあります。このテキスト フィールドは、遷移のトリガと、遷移のガードとアクションを記述します。

トリガとガードは、シグナル受信（入力）シンボルで使用したものと同じ構文に従います。アクションは、アクション（タスク）シンボルと同じ構文に従います。ただし、ダイアグラムのスペースを節約するため、シグナル送信に省略表現を使用します。^s は output s と同じ意味です。

単純な例

```
s1(x) / ^s;
```

ガードのあるシンプル遷移

```
[ x>10 ] / myproc(x);
```

ガードとシグナル受信の両方

```
s1 [ x>10 ] / myproc(x);
```

式

UML の式は、他のほとんどのプログラミング言語の式と類似しています。式には、変数への参照（属性）、リテラル、定数、操作（呼び出し）が含まれます。

ほとんどの式は最後にセミコロンの付いたアクションとして私用されます。たとえば、以下のような式がアクションとして使用されます。

- 代入 式
- 呼び出し式
- new 式

- 条件式

特殊な変数アクセスや複素数値の生成のために使用する式がいくつかあります。

- フィールド式
- インデックス式
- インスタンス式
- this 式

また、変数アクセスと同様に、システムの基礎の動的ステートに依存する式のグループもあり、これらはよく命令式と呼ばれます。

- 任意値（any）式
- now 式
- Pid 式
  - Self
  - Sender
  - Parent
  - Offspring
- タイマー アクティブ式

他にも以下の式が使用できます。

- Assert 式
- 範囲チェック式
- ターゲット コード式

呼び出し式

呼び出し式は操作の呼び出しのために使用されます。操作呼び出しのための実パラメータを含む場合があります。

例 57: 呼び出し式

```
foo(3, true, "mmo")
```

この呼び出し式の値は、操作呼び出し後の戻りパラメータの実際の値です。呼び出された操作が戻りパラメータを持たない場合は、呼び出し式は値を持たず、そのため、式アクション内のスタンドアローン式としてのみ使用できます。

操作呼び出しの前に、実引数として与えられる式は評価されます。ただし、UML は式の評価の順番を定義していないことに注意してください。実際にどの順序で評価が行われるかは、使用するコードジェネレータや生成コードのコンパイラに依存します。したがって、モデルを呼び出し式の実引数の評価の順序に依存しないようにすることを推奨します。

例 58: 引数評価の順序が未定義の場合

```
foo(f1(), f2())
```

この例での操作の呼び出しは、‘f2’、‘f1’、‘foo’（右から左）の順序で行われるか、または ‘f2’、‘f1’、‘foo’（左から右）の順序で行われます。

注記

C コードジェネレータ（AgileC を除くモデルベリファイヤ、モデルエクスプローラを含む）を使用している場合、ターゲットコンパイラの種類によらず、呼び出しの引数は常に左から右に評価されます。UML の標準には評価の順序が定義されていないため、この振る舞いについて独自の解釈をすることは推奨されません。

new 式

new 式には、248 ページの「New」に説明されている new() 構成要素が含まれます。

条件式

条件式は次のような形式です。

```
expr_1 ? expr_2 :expr_3
```

1 番目の式は論理タイプ、2 番目と 3 番目の式は同じタイプです。

まず式 `expr_1` の値が求められます。値が true であれば、`expr_2` の値が求められ、これが条件式の結果となります。値が true でなければ、`expr_3` の値が求められ、これが結果となります。

例 59: 条件式

```
imax = ( i > j ) ? i :j; /* imax = max (i, j) */
```

フィールド式

フィールド式は、構造データ型のフィールド、つまりクラスの属性にアクセスするために使用します。

例 60: フィールド式

```
a.b = true;
test = a.b;
```

インデックス式

インデックス式は、インデックス付きデータ型の要素、通常は配列や文字列にアクセスするために使用します。

例 61: インデックス式

```
iarr[i, j] = 1;
i = iarr[k,l];
```

インスタンス式

インスタンス式は、1 つの操作で複素数値を生成するために使用します。これにより、各フィールドを別個に初期化するのでなく、1 つの操作で構造型を初期化できます。ただし、構造型の初期化にはコンストラクタを使用することを推奨します。

例 62: インスタンス式

```
class sType {
    Integer Age;
    Charstring Name;
    Boolean MaleGender;
}
s = sType(. 'John', 44, true .);
```

インスタンス式は、タグ付き値を含むステレオタイプインスタンスを記述するときにも使用されます。

this 式

this は、現在のインスタンスを指します。this をパッシブ クラスの操作で使用する場合、this はパッシブ クラスのインスタンスを指します。this をアクティブ クラスの操作または状態機械で使用する場合、this はアクティブ クラスのインスタンスを指します。

命令式

命令式には以下の式があります。

- 任意値（any）式
- now 式
- Pid 式
- ステート式
- タイマー アクティブ式

任意値（any）式

any 式は、指定タイプの任意の値を返します。

例 63: any 式

```
anInt = any(Integer);
```

```
output resultSig(any(Boolean));
```

now 式

now 式は、現在の時間値を返します。

例 64: now 式

```
Time time_0 = now;
set(delayTimer, now + 10);
```

Pid 式

Pid 式は、データ型 Pid の式です。Pid 式は、**self**、**parent**、**offspring**、**sender** のいずれかです。

例 65: Pid 式

```
currentClientId = sender;
new serverAgent;
if (offspring != NULL)
  output sender.serverId(offspring)
  else output sender.AllServersBusy;
```

ステート式

ステート式は、現在の状態機械で、最後にいたステートのチェックに使用できます。状態機械に合成状態が含まれる場合、式によって、包含する最も近いスコープの、最後にいたステートが返されます。返される式は、Charstring データ型です。どのステートにもいなかった場合は、空文字列が返されます。

例 66: ステート式

```
if (state  == "idle") return ;
```

タイマー アクティブ式

タイマー アクティブ式は、指定タイマーがアクティブかどうかをチェックするために使用します。論理値が返されます。タイマーは、タイマーがまだタイムアウトしていないか、タイマーがタイムアウトしてもタイマー シグナルがまだ処理（または破棄）されていなければ、アクティブです。

例 67: タイマー アクティブ式

```
if (active(userTimeout)) reset(userTimeout);
```

範囲チェック式

範囲チェック式は、式がランタイム時、値範囲条件に一致するかどうかをチェックするために使用します。形式は次のとおりです。

```
expr_1 in type type_ident
```

type_ident は、制約によってさらに制限が可能です。範囲チェック式は、指定タイプに式が一致するかどうかにより、論理値を返します。

例 68: 範囲チェック式

```
sender in type clientType;
intVar in type Integer constants (1..9, -9..-1);
age in type ageSyntype;
```

ターゲット コード式

ターゲット コード式は、選択された実装言語に依存し、UML パーサによって解析されず、生成されたコードに直接追加される実装言語コードを含みます。

ターゲット コードの形式は次のとおりです。

```
[[ target_code_details ]]
```

ターゲット コード（たとえばインライン C++）には、UML コンテキストで指定されているタイプに一致するものであれば、実装言語の任意の式を含めることができます。

ターゲット コードに次のテキストが含まれている場合

```
]]
```

次のように、# でエスケープする必要があります。

```
#]]
```

ターゲット コードに

```
#
```

が含まれている場合、次のように、# でエスケープする必要があります。

```
##
```

モデル エンティティをターゲット コードから参照する必要がある場合は、形式は #(name) となります。name はモデル中の識別子です。

例 69: ターゲット コード式

```
Real side_a, side_b;
...
```

```
Real hypotenuse = [[ sqrt( pow(#(side_a),2) + pow(#(side_b),2) )]];
```

参照

C Application

合成状態

合成状態は、他のステートや遷移で構成されたステートです。合成状態のサブステートにいる場合、合成状態に定義された遷移のトリガにより、合成状態（およびサブステート）が終了して新規ステートに移ります。

合成状態は、2 つの方法で生成できます : インライン 状態機械定義による、または他の場所で定義された状態機械を参照による方法です。

合成状態は、あるステートについて状態機械図を作成すると暗黙的に作成されます。

合成状態では、ステートシンボルの右上角に「レーキ」シンボルが付きます。

合成状態には、ラベルの付いたエントリ ポイントと終了ポイントを複数持たせることができます。

サブステートの遷移は、外部ステートの遷移よりも優先順位が高くなります。これは、シグナルによってトリガされた遷移と、タイマーによってトリガされた遷移の両方に当てはまります。

これは、UML では、**遷移優先**と呼ばれます。

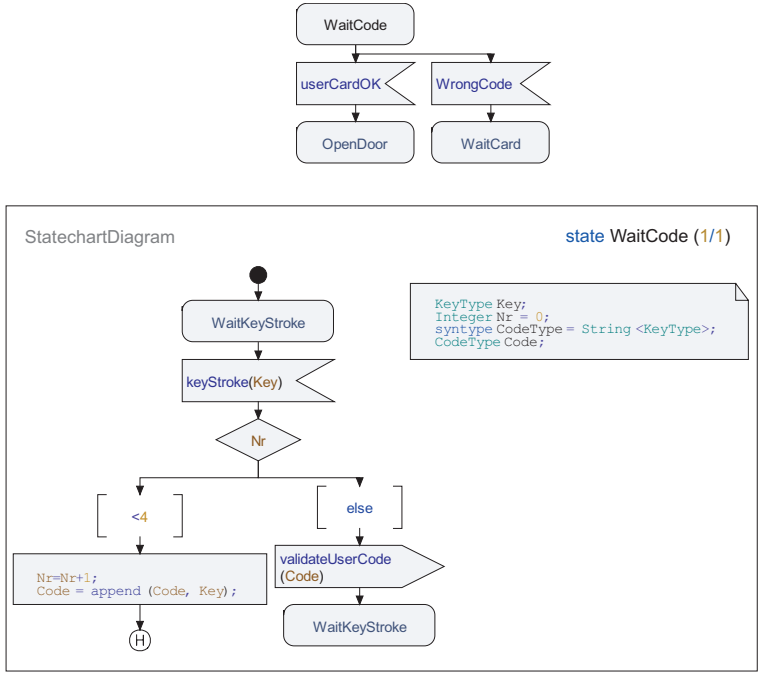

図 98: 合成状態の使用

エントリ接続ポイント

エントリ接続ポイントは、合成状態に入るための名前のある開始点です。エントリ接続ポイントは、合成状態の開始シンボルと、合成状態に入る際に nextstate シンボルで参照されます。

合成状態に、名前のある、または名前のない開始シンボルが少なくとも 1 つ必要です。合成状態では、名前のある開始シンボルは 1 つのみです。

ヒント

エントリ接続ポイントは、モデルビューで定義します。状態機械を選択し、ショートカット メニューから［新規］-［入口接続ポイント］コマンドを選択します。

終了接続ポイント

終了接続ポイントは、合成状態から出るための名前のある終了ポイントです。終了接続ポイントは、合成状態のリターン シンボルと、合成状態から出る接続遷移で参照されます。

合成状態から出る接続遷移が複数ある場合、これらの接続遷移のうち 1 つは名前がなくてもかまいません。

ヒント

終了接続ポイントは、モデルビューで定義します。状態機械を選択し、ショートカット メニューから［新規］-［出口接続ポイント］コマンドを選択します。

状態機械継承

状態機械は、状態機械間の継承によって直接特化するか、状態機械を所有するアクティブ クラスの特化により、特化できます。特化された状態機械では、元の状態機械に特性を追加したり、特性を変更したりできます。追加できる特性は、ステート、遷移、変数、その他状態機械で宣言できるエンティティです。特化によって特性が変更されるようにするには、元の状態機械で**仮想**と宣言されていなければなりません。仮想定義は、特化された状態機械で再定義できます。状態機械で、以下の概念を仮想（したがって再定義可能）にできます。

- 遷移
  - 開始
  - シグナル受信
  - ガード
  - 保存
- 操作

操作本体

操作本体は、ステートのないメソッドです。アクションは、多くの場合、他のアクションのリストを含んだ複合アクションになります。

つまり、実行方法の定義を UML 言語で形式的に表現するのではなく、他の言語を使用することが可能です。その場合、操作本体には、非形式的記述を含む非形式的表現が含まれます。

参照

状態機械実装

インターナル

実装

状態機械実装

状態機械実装は、ステートと、状態機械 シグニチャの実現化に必要な他のすべてのものを含むメソッドです。状態機械実装は、通常、状態機械定義時に暗黙的に定義されます。

参照

状態機械

インターナル

実装

インターナル

インターナルは、クラス定義をシグニチャ指向パート 1 つと実装指向パート 1 つに分割し、クラスのシグニチャを、クラスの実装とは別のファイルに格納できるようにするために使用します。この目的は、バージョン ハンドリングやデリバリーをシグニチャと実装で別個にし、コンポーネント ベースのモデリングを容易にすることです。

参照

状態機械実装

操作本体

実装

テキスト拡張シンボル

テキスト拡張シンボルをアクション シンボルに接続して、アクション シンボルの内容を表示できます。これは、遷移指向のフローで、大量のテキストをもつアクションによってダイアグラムの概要がわかりにくくなる場合などに特に役立ちます。アクション コードは、アクション シンボルまたはテキスト拡張シンボル内で編集可能です。

デプロイメント モデリング

デプロイメント モデリングでは、システムのランタイム アーキテクチャのモデリングを行います。配置可能なソフトウェア、アーティファクトが、物理情報処理リソースを表すノードにどのように配置されるかを記述します。デプロイメント スペシフィケーションは、アーティファクトがどのようにノードに配置されるかを記述するのに使用します。関連は、ノード間の接続のモデリングに使用します。

配置図

配置図は、相互接続された一連のアーティファクトに配置された一連のノードを指定します。デプロイメント スペシフィケーションは、アーティファクトをノードに配置する際に使用する実行パラメータの指定に使用します。実行環境を使用して、一連のサービスを、配置されているアーティファクトに提供するノードのモデリングができます。

例

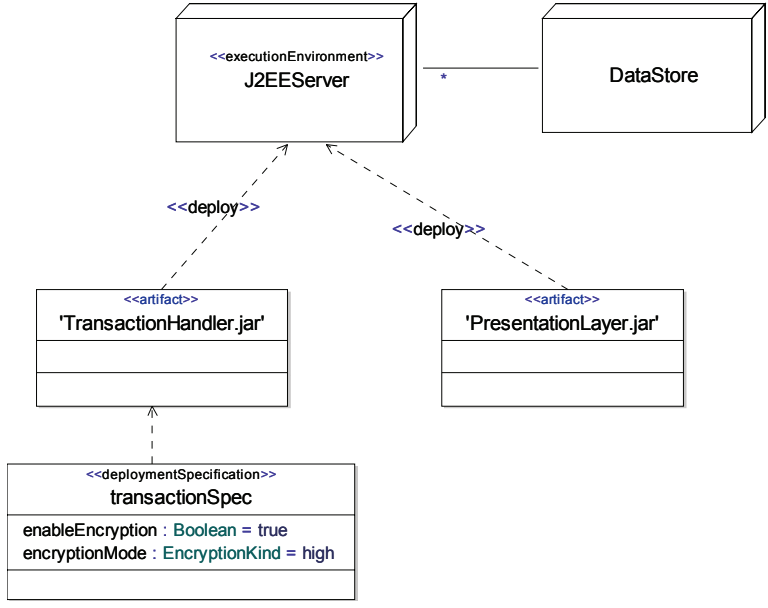

図 99: 配置図

配置図のモデル要素

配置図には、以下の要素が使用されます。

- アーティファクト
- ノード
- 実行環境
- デプロイメント スペシフィケーション
- アーティファクト
- クラス
- 関係

参照

クラス図

コンポーネント図

アーティファクト

アーティファクトは、ソフトウェア開発プロセスで使用または生成される物理情報を表します。ソース ファイル、スクリプト、ライブラリ、実行形式プログラムは、アーティファクトの例です。

アーティファクトは、表現関係によって複数の要素を表します。つまり、アーティファクトは、これらの要素から組み立てられます。たとえば、C++ のヘッダー ファイルを表すアーティファクトには、ヘッダー ファイルで宣言されたクラスとの表現関係を持たせることができます。この情報は、モデルから物理ヘッダー ファイルを生成する際、コード ジェネレータで使用できるようになります。

デプロイメント モデリング中、アーティファクトは、デプロイメント関係を使用してノードに配置されます。

アーティファクトはクラスと似ており、属性と操作を持たせることができます。アーティファクトは、（任意の要素の）依存、（アーティファクト間の）汎化、（通常は他のアーティファクトの）合成の各関係に関与することもできます。また、アーティファクトは名前空間であり、他のモデル要素を所有することもできます。

シンボル

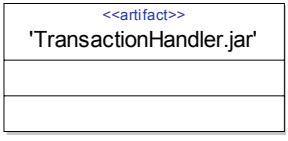

図 100: アーティファクト シンボル

アーティファクト シンボルは、クラス シンボルと同じです。ただし、`<<artifact>>` というキーワードを上部に追加します。

ノード

ノードは、名前のある情報処理リソース、通常は特定のコンピュータです。ノードは、関連を使用して、モデル ネットワーク トポロジーと接続できます。

シンボル

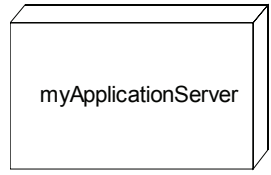

図 101: ノード シンボル

構文

ノードは、内部に名前のある三次元立方体として描かれます。

実行環境

アーティファクトの実行環境が配置された特殊な種類のノードです。実行環境は、通常、実行中にアーティファクトが必要とする一連のサービスで構成されます。

J2EE bean を配置するために準備された J2EE サーバがその典型的な例です。

シンボル

図 102: 実行環境シンボル

構文

ノードと同じです。ただし、<<executionEnvironment>> ステレオタイプが適用されます。

デプロイメント スペシフィケーション

デプロイメント スペシフィケーションは、ノードへの配置時、アーティファクトの実行パラメータとして機能する一連のプロパティを指定するために使用します。

ディプロイメント スペシフィケーションは、仕様からアーティファクトへの依存を示すことにより、アーティファクトに適用します。

シンボル

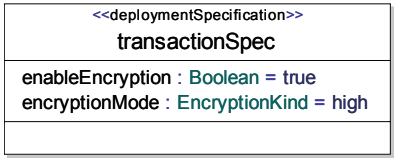

図 103: デプロイメント スペシフィケーション シンボル

構文

クラスと同じです。ただし、<<deploymentSpecification>> ステレオタイプが適用されます。

関係

配置図では、以下の関係を使用できます。

- デプロイメント
- 表現
- 関連
- 集約
- 合成
- 汎化
- 依存

デプロイメント

アーティファクトを、デプロイメント ターゲット、通常はノードに配置するために使用する特殊な種類の依存です。ノードに配置されたアーティファクトは、そのノードのコンテキストで実行されます。

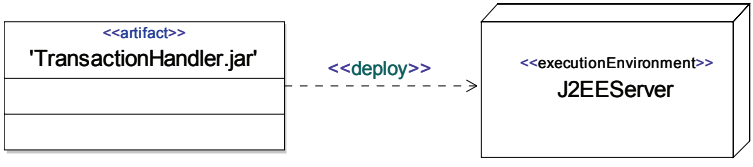

図 104: デプロイメント依存

表現

表現は、アーティファクトから他の一連の要素に使用して、アーティファクトがそれらの要素から組み立てられていることを記述する特殊な種類の依存です。

たとえば、C++ のヘッダー ファイルを表すアーティファクトには、ヘッダー ファイルで宣言されたクラスとの表現関係を持たせることができます。この情報は、モデルから物理ヘッダー ファイルを生成する際、コード ジェネレータで使用できるようになります。

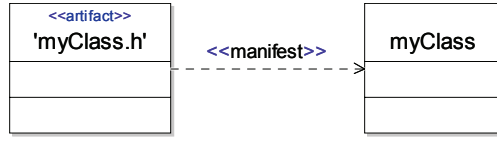

図 105: 表現依存

UML の関係

ラインの編集に関する一般的なヘルプについては、以下を参照してください。

119 ページの「ラインの描画」

120 ページの「ラインの移動」

120 ページの「ラインの削除」

120 ページの「ラインの方向変更と双方向化」

依存

依存は 2 つの定義間の関係で、何らかの理由で、一方の定義（*クライアント*）が、もう一方の定義（*サプライヤ*）に依存していることを示します。依存のセマンティックは比較的緩やかなので、他の関係クラスでは不適切になってしまい一定の関係のモデリングができない場合でも使用できます。

依存の特徴的な使用法が 1 つあります。アクティブ クラスのインスタンス間の生成関係、つまり、あるインスタンスが New 文を使用することによって、あるクラスの新規インスタンスを生成する関係を示す場合です。この場合、パート間、またはパートと振る舞いシンボル（包含するアクティブ クラスの状態機械を参照している）の間で依存を使用できます。

ステレオタイプを適用することによって、依存に対してより詳細な意味を与えるのが一般的です。インポートとアクセス依存を参照してください。

汎化

汎化は 2 つのシグニチャ（クラスや操作など）間の関係で、一方がより一般的なシグニチャで、もう一方がより特化されたシグニチャであることを表現します。特化されたシグニチャは、一般的なシグニチャのメンバー定義を継承し、さらに追加のメンバーを含む場合もあります。このため、汎化関係は「継承」とも呼ばれます。

2 つのタイプ（2 つのクラスなど）間で汎化が行われると、より特化されたタイプが、より一般的なタイプ（スーパータイプとも呼ばれる）のサブタイプを定義します。これは、より一般的なタイプのインスタンスを、より特化されたタイプのインスタンスと置き換えることができるということです。他の言い方をすると、特化された型は、割り当てにおいて、一般的な型を置き換えられます。

構文

汎化ラインには、弁別子を含むテキスト フィールドがあります。

実現化

実現化関係は、特殊な汎化関係です。実現化は、クラスとインターフェイスの間で使用され、実現化するクラスがインターフェイスに一致する (インターフェイスを実装する) ことを表します。

関連

関連は 2 つ以上の分類子のセマンティック関係で、これらの分類子のインスタンスが関係付けられることを示します。

シンボル

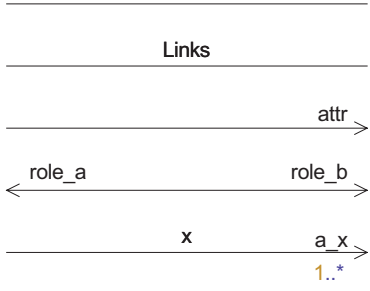

図 106: 関連

ラインには、名前フィールドが 1 つ、役割名フィールドが 2 つ、多重度フィールドが 2 つ含まれます。

関連には関連の端が 2 つあり、これらは属性として表されます。これらの属性は、両方とも関連が所有する（関連付けられている分類子のいずれも関連に影響を受けない状況を反映）か、関連が一方の属性を所有し、もう一方を 1 つの接続分類子 C が所有する（関連が C からの方向のみに誘導可能である状況を反映）か、各接続分類子 C が属性それぞれを所有する（関連が両方向に誘導可能である状況を反映）場合があります。関連が一方向の場合は、2 つ目の（リモート）属性は、必要な場合（役割名や多重度を持っている場合など）のみ存在します。

関連には、関連自体に所属し、特定の関連の端には所属しないプロパティを持たせることができます。

関連は、両方向に誘導可能です。

多重度

関連の端の多重度は、関連によって関係付けができるクラスのインスタンスの数を定義します。

集約の種類

関連は、通常の関連、つまり集約、または合成です。

ラインの端部パートをクリックすると表示されるショートカット メニューで、集約タイプを変更できます。選択肢は、[関連]、[集約]、[合成] です。集約タイプを選択するには、まず役割名を追加する必要があります。

- 集約ラインは、集約クラスのインスタンスが、非形式的に、コンポーネント クラスのインスタンスに所有されていると見なされることを指定します。
- 合成ラインは、コンポーネント クラスが存在する間だけ集約クラスのインスタンスが存在する、強力な集約形式を指定します。したがって、包含されるインスタンスのライフタイムは、それを包含するインスタンスのライフタイムと強く関連しています。

誘導可能端

誘導可能端は、もう一方の端のタイプである分類子の属性でもある関連の端です。

シンボル

ラインには、名前フィールドが 1 つ、役割名フィールドが 2 つ、多重度フィールドが 2 つ含まれます。

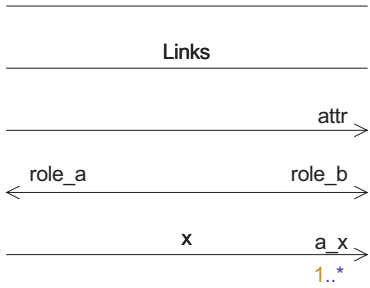

図 107: 関連

例

例 70: ロール テキスト

```
+ myrole
```

例 71: 多重度テキスト

無限範囲

```
*
```

範囲条件

```
0..3
```

複数の範囲条件

```
1..7,>10
```

参照

属性
集約
合成

集約

集約は、特殊な種類の関連です。集約関係（全体／部分関係）を指定する 2 項関連です。

集約には、集約端と部分端という 2 つの端があります。集約は、集約端の分類子のインスタンスが、部分端の分類子のインスタンスを集約することを指定します。集約インスタンスは、別の集約の一部になることもできます。

集約部分は、複数の集約の一部になることができます。

シンボル

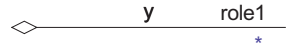

図 108: 集約

参照

属性

関連

合成

合成

合成は、特殊な種類の集約です。合成パートは、合成によって所有され、1 つの合成の一部にしかなることができません。

タイプがアクティブ クラスの合成パートは、合成構造図で記述されている、クラスの内部構造のパートとしても使用できます。

シンボル

図 109: 合成と対応属性

参照

属性

関連

集約

パート

合成構造図

包含

包含関係は、1 つの定義が別の定義を含んでいる状況を表現します。含まれる定義は、コンテナ定義のスコープに表示されます。この関係が名前空間の間で使用される場合には、名前空間のネストとも呼ばれます。

シンボル

包含ラインは「含む側」定義から「含まれる側」定義に向かって描画されます。「含む側」サイドにはプラスマークが表示されます。

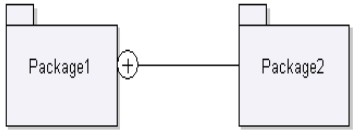

図 110: 包含

拡張

拡張は、ステレオタイプとメタモデル クラスの間で使用され、ステレオタイプがメタクラス (メタモデル クラス) を拡張することを示します。

関連

説明

関連の詳細については、ユース ケース モデリングセクションを参照してください。

参照

共通シンボル

フレーム

ダイアグラムのシンボルは、キャンバスに置かれたフレーム シンボルで囲まれます。

- フレームにはすべての辺にマージンがあります。

- フレームは、マージンを含め、キャンバスのすべての方向にサイズ変更や移動ができます。

テキスト シンボル

テキスト シンボルは、変数、インターフェイス、データ型などの定義に使用します。

シンボルにラインを接続することはできません。

構文

例 72: インターフェイスとシンタイプの定義を含む

```
interface i {
  signal s;
}
syntype s = Integer;
```

例 73: ステレオタイプの決まったクラスの定義を含む

```
<<struct>> class X {
  private Integer I;
  void inc ( Integer incr ) {
    I = I + incr;
  }
}
```

コメント

コメント シンボルは、ダイアグラムのグラフィック シンボルに関するコメント テキストの定義に使用します。

コメントは、テキスト構文で作成することもできます。

コメント シンボル

コメントシンボルはテキスト シンボルと同様に描くことができますが、シンボルの左上端に読み取り専用を表すマークが表示されます。マークは「//」であり、これにより制約シンボルなどと区別されます。注釈ラインを使用して、シンボルを別のシンボルに接続できます。

コメント シンボルは左側で接続されますが、ショートカット メニューでシンボルを水平に反転させて、右側から接続することもできます。ダイアグラム内のコメント シンボルが他のシンボルに接続されない場合、コメント モデル要素はダイアグラムを所有している要素に属します。コメント シンボルがダイアグラム内の 2 つ以上のシンボルに接続される場合、コメント モデル要素はダイアグラムを所有している要素に属します。

構文

テキストは非形式的であり構文的にチェックされません。

参照

コメントの処理

制約

制約シンボルを使用して、ダイアグラム内のグラフィック シンボルに関連する制約テキストを定義できます。

制約はテキスト構文で作成することもできます。

制約シンボル

制約シンボルはコメントシンボルと同じように作成されますが、シンボルの左上隅には読み取り専用テキスト ラベルがあります。テキストは “{}” に設定されて、制約シンボルとコメント シンボルが区別されます。注釈ラインを使用して、シンボルを別のシンボルに接続できます。

構文

テキストは非形式的であり、構文的にチェックされません。

ステレオタイプ インスタンス

ステレオタイプ インスタンス シンボルを使用して、モデル要素に関連するステレオタイプ インスタンス テキストを定義できます。

ステレオタイプ インスタンス シンボルは、テキスト構文で作成することもできます。

ステレオタイプ インスタンス シンボル

ステレオタイプ インスタンス シンボルはコメントシンボルと同じように作成されますが、シンボルの左上隅には読み取り専用テキスト ラベルがあります。テキストは「<<>>」に設定されて、制約シンボルとコメント シンボルが区別されます。注釈ラインを使用して、シンボルを別のシンボルに接続できます。

構文

テキストは非形式的であり、構文的にチェックされません。

注釈ライン

注釈ラインは、コメント、制約、およびステレオタイプ インスタンス シンボルを別の要素に接続します。

注釈ラインはシンボルのライン ハンドルから引き、ダイアグラム フレーム内の他の任意のシンボルにつなぐことができます。ただし、コメント、制約、ステレオタイプ インスタンス シンボル、およびテキスト シンボルにはつなぐことはできません。

拡張性

UML は、一定の管理の下でカスタマイズ可能な言語です。UML 構成要素を拡張したり、特定の目的で使用するために特化するために、あらかじめ決められた仕組みがあります。

UML の拡張性は、プロファイル とメタモデルの概念に基づいています。

メタモデルは、ツールのリポジトリに格納する情報の記述に使用する、特殊な種類の UML パッケージ クラス モデルです。パッケージは、パッケージ名の前に <<metamodel>> というキーワードが付いていればメタモデルです。通常、メタモデルは、メタクラスを定義する <<metaclass>> というキーワードによってステレオタイプの決められた一連のクラスを含みます。

別のメタモデルを定義し、これらのメタモデルに基づき、内蔵リポジトリを使用してユーザーレベル モデルを格納することもできます。唯一の要件として、メタモデルは、ランタイム リポジトリとストレージの定義に使用されるオブジェクト モデルにマッピング可能でなければなりません。

プロファイルは、パッケージ名の前のヘッダーに <<profile>> というキーワードの付いた特殊な種類のパッケージです。プロファイルには、属性（*タグ付き値定義*と呼ばれる）を持ち、1 つまたは複数の*メタクラス*を拡張する一連の*ステレオタイプ*が含まれます。

ユーザー モデルでは、ステレオタイプを拡張メタクラスのインスタンスであるオブジェクトに適用できます。これで、値の追加が可能になります。

メタモデル

メタモデルは、モデル リポジトリに格納される情報の概念的ビューを定義する一連のメタクラス、メタ属性などです。メタモデルは主に、プロファイル定義の基礎を形成するために使用します。

ユーザー プロファイルにより、より多くの情報をモデル要素に関連付けるためにメタクラスを拡張するステレオタイプが定義できます。追加分の情報は、ユーザーから見て、プロパティ エディタを使用して編集可能で、モデル リポジトリに格納されます。

UML ツール セットにより、特定のモデルのさまざまなビューを提供する複数のメタモデルを表現できます。

ヒント

メタモデルの例が参照できるようになっています。［モデル ビュー］の［Library］フォルダの TTDMetamodel パッケージをチェックしてください。このパッケージは、格納されている情報を記述する単純なメタモデルです。TTDMetamodel の目的は、基礎リポジトリ構造によく似たビューを提供することです。このメタモデルの各クラスは、リポジトリ定義のコア クラスに直接対応しています。だだし、TTDMetamodel は、ステレオタイプに役立つクラスのみが含まれているという意味で、コア レポジトリを単純化したものです。コア レポジトリ モデルのほとんどすべての関連と属性が省略されているという点でも、単純化されています。

メタクラス

メタクラスは、UML リポジトリに格納されている一連の要素のカテゴライズに使用します。メタモデルで、クラス名のステレオタイプを <<metaclass>> としたクラス シンボルを使用して定義できます。

ステレオタイプ

ステレオタイプは、所定エンティティのモデルに格納される情報の拡張に使用します。追加の情報は、ステレオタイプの属性で記述します。

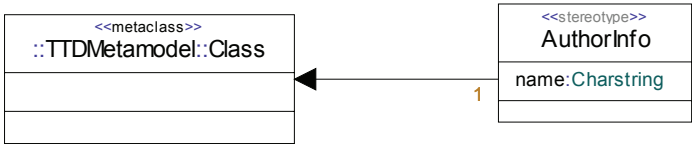

図 111: ステレオタイプの例

274 ページの図 111 は、すべてのクラスを、クラス定義の作者に関する情報で拡張する例です。これは、メタクラス *TTDMetamodel::Class* を拡張する属性 *name* を持つステレオタイプ *AuthorInfo* を定義することによって行うことができます。

タグ定義

タグ定義は、ステレオタイプの属性です。ステレオタイプを適用すると、特定の値を与えることにより、タグ定義が使用されます。

タグ付き値

タグ付き値は、タグ定義に使用できる値です。値は、プロパティ エディタを使用して設定します。

参照

拡張

プロファイル

プロファイルは、ステレオタイプが <<profile>> の特殊な種類のパッケージです。プロファイルは、メタクラスを拡張するステレオタイプを定義することにより、UML リポジトリに格納できる情報を拡張するのに使用します。274 ページの図 111 は、単純なプロファイルの例です。

プロファイルは、パッケージ インポートまたはアクセス構成要素を使用して適用します。たとえば、モデルに何らかのプロファイルを適用する場合、モデルの上部パッケージには、該当のプロファイルを定義するパッケージを参照するインポートまたはアクセスが必要です。

拡張

拡張は、ステレオタイプとメタモデル クラスの間で使用され、ステレオタイプがメタモデル クラスを拡張することを示します。

拡張ラインに関連付けられたテキスト フィールドが 1 つあります。このフィールドに入るテキストは、1 または 0..1 です。テキストが 1 の場合、拡張メタクラスのインスタンスであるすべての要素に、ステレオタイプが自動的に適用されます。

テキストが 0..1 の場合は、ステレオタイプを手動で適用する必要があります。274 ページの図 111 は、拡張ラインの例です。

ステレオタイプを手動で適用すると、一部のシンボル（クラス シンボル、シグナル シンボルなど）には、適用されたステレオタイプが示されます。

定義済みデータ

UML のデータ モデリング構成要素は強力で、さまざまな方法によるモデリングやデータ定義に対応します。ただし、UML 内蔵のデータのタイプは多くありません。アプリケーションドメインによって UML をさまざまなデータのタイプで拡張できるようになっています。これは、モデル ライブラリ（定義済みパッケージとも呼ばれることも多い）でデータ型を定義することによって行います。

定義済み

このパッケージには、常に使用できる操作を持つ汎用データ型が含まれています。

参照

データ型

定義済み

Predefined 定義済みパッケージは、独自の UML 拡張で、プロジェクトで常に使用できます。このパッケージは、プロジェクトで定義されたモデルによって自動的に使用されます。パッケージにより、いくつかのデータ型、その他ユーティリティが定義されます。

一部のデータ型は OMG UML にあります（Integer、Boolean など）が、この定義済みパッケージには、通常はないデータ型の操作があります。

データ型ごとに、タイプの表現に適用する一連の操作があります。

パッケージには、以下の定義が含まれます。

種類	定義
データ型	Boolean, Character, String, Charstring, Integer, Natural, Real, Array, Any
定数	PLUS_INFINITY, MINUS_INFINITY

Predefined パッケージは、［モデル ビュー］で直接確認したり参照したりできます。各プロジェクトに、定義済みパッケージのノードがあります。このノードを拡大すると、使用できるデータ型、演算子、その他の定義が参照できます。

PLUS_INFINITY

PLUS_INFINITY はデータ型 Real の定数です。ホスト、または特定のターゲットで使用できる最大 Real 数を参照するために使用できます。

MINUS_INFINITY

PLUS_INFINITY はデータ型 Real の定数です。ホスト、または特定のターゲットで使用できる最大の負の Real 数を参照するために使用できます。

メタモデル クラス

以下に、他の重要なメタモデル クラスについて説明します。

メタモデル プロファイル

TTDMetamodel は、［モデル ビュー］で直接確認したり参照したりできます。プロジェクトを追加する場合、プロファイルが適用されたモードが常にあります。TTDMetamodel はこれらのプロファイルの 1 つです。ライブラリ ノードと TTDMetamodel ライブラリ パッケージを拡大すると、言語モデル要素、抽象メタクラス、その間の関係が参照できます。

分類子

分類子は、UML 言語のメタクラスです。

分類子はデータの記述で、インスタンスの集合、つまりインスタンス セットのシグニチャです。分類子はタイプを定義します。たとえば、StructuralFeature タイプなどです。分類子は、関連により、他の分類子と関連付けることができます。

ほとんどのクラス的なモデル要素は分類子です。以下のようなものがあります。

- クラス
- データ型、シンタイプ、選択
- ステレオタイプ
- インターフェイス
- コラボレーション

シグニチャ

シグニチャは、UML 言語のメタクラスです。

シグニチャは、別のシグニチャの定義の基礎にできるエンティティです。これを可能にする主な仕組みは以下の 2 つです。

- 特化、または継承
- パラメタライゼーション

特化とは、スーパー シグニチャを一連のサブ シグニチャに特殊化できるということです。各サブ シグニチャは、スーパー シグニチャのすべてのプロパティを継承し、他にもプロパティを持つことができます。メタモデルでは、特化の仕組みは、シグニチャが所有する汎化クラスによってモデリングされます。

パラメタライゼーションとは、シグニチャに仮コンテキスト パラメータのリストを持たせることができるということです。このようなシグニチャはテンプレートと呼ばれます。テンプレートの仮コンテキスト パラメータは、テンプレートのインスタンス化時（TemplateTypeInstantiation などで）に実コンテキスト パラメータと置き換えることができます。パラメタライゼーションにより、シグニチャがより柔軟になり、さまざまなコンテキストで使用できます。メタモデルでは、パラメタライゼーションの仕組みは、シグニチャが所有する ContextParameter クラスによってモデリングされます。

シグニチャに基づいて新規シグニチャを定義するこれらの 2 つの仕組みに加え、シンタイプという、シグニチャ 1 つのみを持つ仕組みがあります。この仕組みでは、シグニチャの制約により、新規シグニチャを定義します。

シグニチャには、実装を持つものもあります。その場合、シグニチャは実装のファサードとして機能し、シグニチャのユーザーが知らなくてもよい詳細をすべて非表示にします。ファサードにより、実装から定義が分離され、システムのパートのコンパイルが個別にできるようになります。C プログラミングのヘッダー ファイルの使用などと比較してください。ファサードには、以下のことが当てはまります。

- ファサードは実装に依存しない

- ファサードは使用方法に依存しない（これはすべての定義について当てはまります。

実装がファサードに依存します。

以下のモデル要素はシグニチャです。

- 分類子
- 操作、シグナル、タイマー

実装

実装は、シグニチャのユーザーが知る必要はないが、実行には必要な、シグニチャに関する詳細を記述します。通常、シグニチャはエンティティの静的プロパティを記述します。対応する実装は、動的プロパティのほうに関係します。

実装には、インターナルとメソッドの 2 種類があります。インターナルは、クラスの構造を、物理的に、または通信の観点から記述します。メソッドは、操作、StateType、クラスをランタイム実行の観点から記述します。

実装はシグニチャ（ファサードとも呼ばれる）にのみ依存し、シグニチャの使用方法には依存しません。これは、システムのパートの個別分析ができるようにするために重要です。

メソッド

メソッドは、操作の実装です。ランタイム時にどのように実行されるかを記述します。メソッドには 3 種類あり、それぞれに実行セマンティックがあります。

- 操作本体：操作本体のアクションの実行により実行されるステートレス メソッドです。
- 状態機械実装：アクティブ ステートで開始できる遷移に関連するアクションを実行することによって実行される、ステートと遷移のあるメソッドです。
- 相互作用：一連の属性間の相互作用と情報のやりとりを記述するメソッドです。他のメソッドと異なり、相互作用は、実行方法を完全に指定するだけでなく、実際どのように実行されるかを記述したり（トレースの記述）、どのように実行しなければならないかを部分的に記述したり（これで他のメソッドにセマンティック要件を適用できます）するためにも使用できます。
- アクティビティ実装：小さい振る舞い単位の管理セットを実行するメソッドです。

シグニチャと実装

シグニチャと実装は、UML 言語のメタクラスです。シグニチャはエンティティを宣言し、実装は同じエンティティを定義します。すなわち、これらの概念により、シグニチャを実装から物理的に分離できます（C や C++ のヘッダーファイルと比較してください）。

これが可能な概念を以下に示します。

操作

操作シグニチャおよび操作本体、アクティビティ実装、状態機械実装または相互作用

アクティビティ

アクティビティ シグニチャとアクティビティ実装

状態機械

状態機械 シグニチャと状態機械実装

クラス

クラス シグニチャとインターナル

注記合成合成合成

SysML の要求部分については、要求プロファイルを使用しています。詳細については、要求のモデリングを参照してください。

-
-
-

参照

要求レポート

参照

要求レポート

将来サポートされなくなる概念

SysML の仕様は現在も進展中であり、すでに実装された機能でも変更を余儀なくされる可能性があります。実際すでにいくつかの機能はサポートされなくなる予定になっています。

サポートされなくなる対象は `SysMLDeprecated` という名前のパッケージに移動されました。これらの使用は推奨できません。このパッケージ内の要素 (およびステレオタイプインスタンスのような任意のインスタンス) は将来のリリースにおいて削除されます。

これらの概念のデータを喪失しないためには、手動または API を使用したプログラミングによってデータを移動してください。

変更または非推奨となった概念は以下のとおりです :

- verifyMethodKind
- riskKind
- optimizationDirectionKind
-

スケジューラビリティ、パフォーマンス、時刻のプロファイル

このセクションでは、UML Profile for Schedulability、Performance、および Time（別名 UML Real -time profile）のすべてのステレオタイプ、タグ付き値、列挙をリストアップします。

注記

タグ付き値の一部は、テキスト構文を使用した編集のみで可能です。このようなタグ付き値については、本文ではイタリックで表記しています。

RTresourceModeling

GRMacquire

GRMblocking : Boolean

GRMcode

GRMrealize

GRMmapping : GRMmappingString

GRMdeploys

GRMrelease

GRMrequires

RTtimeModeling

RTaction

RTstart : RTtimeValue

RTend : RTtimeValue

RTduration : RTtimeValue

RTclkInterrupt

RTstimulus

RTstart : RTtimeValue

RTend : RTtimeValue

RTclock

RTclockId : Charstring

RTdelay

RTevent

RTat : RTtimeValue

RTinterval

RTintState : RTtimeValue

RTintEnd : RTtimeValue

RTintDuration : RTtimeValue

RTnewClock

RTnewTimer

RTtimerPar : RTtimeValue

RTpause

RTreset

RTset

RTtimePar : RTtimeValue

RTstart

RTtime

RTkind : RTkindEnum

RTtimeout

RTtimer

RTduration : RTtimeValue

RTperiodic : Boolean

RTtimeService

RTtimingMechanism

RTstability : Real

RTdrift : Real

RTskew : Real

RTmaxValue : RTtimeValue

RTorigin : Charstring

RTresolution : RTtimeValue

RToffset : RTtimeValue

RTaccuracy : RTtimeValue

RTcurrentVal : RTtimeValue

RTkindEnum

リテラル：

- dense
- discrete

RTconcurrencyModeling

CRaction

CRatomic : Boolean

CRasynch

CRconcurrent

CRcontains

CRdeferred

CRimmediate

CRthreading : CRthreadingEnum

CRmsgQ

CRsynch

CRthreadingEnum

リテラル：

- local
- remote

SAprofile

SAaction

SApriority : Integer

SAblocking : RTtimeValue

SAdelay : RTtimeValue

SApreempted : RTtimeValue

SAready : RTtimeValue

S*Arelease : RTtimeValue*

SAworstCase : RTtimeValue

SAabsDeadline : RTtimeValue

SAlaxity : SAlaxityEnum

SArelDeadline : RTtimeValue

SAengine

SAaccessPolicy : SAaccessControlPolicyEnum

SAcontextSwitch : TimeFunction

SAschedulable : Boolean

SApreemptible : Boolean

SApriorityRange : Range

SArate : Real

SAschedulingPolicy : SAschedulingPolicyEnum

SAutilization : Real

SAaccessPolParam : Real

SAowns

SAprecedes

SAresource

SAacquisition : RTtimeValue

SAcapacity : Integer

SAdeacquisition : RTtimeValue

SAconsumable : Boolean

SAaccessControl : SAaccessControlPolicyEnum

SAptyCeiling : Integer

SApreemptible : Boolean

SAaccessCtrlParam : Real

SAresponse

SAutilization : Real

SAspare : RTtimeValue

SAslack : RTtimeValue

SAoverlaps : Integer

SAschedRes

SAscheduler

SAschedulingPolicy : SAschedulingPolicyEnum

SAsituation

SAtrigger

SAschedulable : Boolean

SAoccurrence : RTarrivalPattern

SAendToEnd : Charstring

SAusedHost

SAuses

SAlaxityEnum

リテラル：

- hard
- soft

SAschedulingPolicyEnum

リテラル：

- rateMonotonic
- deadlineMonotonic
- HKL
- fixedPriority
- minimumLaxityFirst
- maximizeAccruedUtility
- MinimumSlackTime

SAaccessControlPolicyEnum

リテラル：

- FIFO
- priorityInheritance
- noPreemption

- highestLockers
- priorityCeiling

PAprofile

PAclosedLoad

PArespTime : PAperfValue

PApriority : Integer

PApopulation : Integer

PAextDelay : PAperfValue

PAcontext

PAhost

PAutilization : Real

PAschdPolicy : PAschdPolicyEnum

PArate : Real

PActxtSwT : PAperfValue

PAprioRange : Range

PApreemptable : Boolean

PAthroughput : Real

PAopenLoad

PArespTime : PAperfValue

PApriority : Integer

PAoccurrence : RTarrivalPattern

PAresource

PAutilization : Real

PAschdPolicy : PAschdPolicyEnum

PAcapacity : Integer

PAaxTime : PAperfValue

PArespTime : PAperfValue

PAwaitTime : PAperfValue

PAthroughput : Real

PAstep

PAdemand : PAperfValue

PArespTime : PAperfValue

PAprob : Real

PArep : Integer

PAdelay : PAperfValue

PAextOp : PAextOpValue

PAinterval : PAperfValue

PAschdPolicyEnum

リテラル：

- FIFO
- priority

RSAprofile

RSAclient

RSAtimeout : RTtimeValue

RSAclPrio : Integer

RSAprivate : Integer

RSAconnection

RSAshared : Boolean

RSAhiPrio : Integer

RSAloPrio : Integer

RSAmutex

RSAorb

RSAserver

RSAsrvPrio : Integer

RSAchannel

RSAschedulingPolicy : RSAschedulingPolicyEnum

RSAaverageLatency : RTtimeValue

RSAschedulingPolicyEnum

リテラル：

- FIFO
- RateMonotonic
- DeadlineMonotonic
- HKL
- FixedPriority
- MinimumLaxityFirst
- MaximizeAccruedUtility
- MinimumSlackTime

5

エラー メッセージと警告メッセージ

このドキュメントは、UML ツール セットから表示されるエラー メッセージと警告メッセージのリファレンス ガイドとしてご利用いただけます。

一般的なアプリケーションのエラーと警告

DOORS Analyst の minidump ファイル （Windows）

DOORS Analyst には、Windows プラットフォームの機能を取り込む デバッグ情報が組み込まれています。実行中に DOORS Analyst が強制終了されて minidump ファイルが作成されたことを示すメッセージが表示されたら、最寄りの Telelogic サポートにご連絡ください。minidump ファイルには、現在の呼び出しスタックが含まれており、どの呼び出しが実行されたか特定するのに役立ちます。内部のツール呼び出しが原因でエラーが発生したのかどうかを判断するため、まだ特定されていない問題を解決できることもあります。また OS 上の依存関係と第三者呼び出しを考慮するので、物理的な外部環境とのインテグレーションを改善し、DOORS Analyst が依存している他社のソフトウェアの要件を明確にできます。

図 1: 強制終了時のメッセージボックス

Minidump ファイル の場所

minidump ファイルは、デフォルトでローカル設定ディレクトリに作成されますが、環境変数を使用して、格納場所を変更することもできます。

デフォルトの格納場所

```
C:¥Documents and Settings¥<user>¥Local Settings¥Temp
```

環境変数を使用して格納場所を変更する例

```
TAU_DUMP_PATH=c:¥DevTools¥Telelogic¥minidumps¥
```

Minidump ファイルの内容

minidump ファイルには呼び出しスタックとレジスタのみが含まれ、メモリはありません。つまり、minidump ファイルが作成された元のモデルに関する情報は含まれません。

エラーと警告

フェーズと識別子

UML モデルを別の言語または形式に変換するプロセスには複数のフェーズがあります。モデルの処理中に、問題が起こった場所を識別できるようフェーズごとにエラーメッセージと警告メッセージが表示される場合があります。フェーズを識別する接頭辞は以下のとおりです。

- TSX：構文分析
- TSC: セマンティック チェック
- TNR: 名前解決

TSX：構文分析

構文分析では、UML による構築を正しく行えるように言語要素がどのように構築されて集成されているかをチェックします。

TSC: セマンティック チェック

セマンティック チェックでは、UML モデルが完全で、言語構成要素間の関係に意味があることをベリファイします。

TNR: 名前解決

名前解決では、UML エンティティの名前を識別して、モデル内の正しい定義へのバインドを試行します。

TSX：構文分析

構文分析では、UML による構築を正しく行えるように言語要素がどのように構築されて集成されているかをチェックします。

構文エラーのほとんどの直接的要因は、UML モデルで特定できます。

このフェーズで発生するエラーと警告には、TSX という接頭辞が付きます。

```
Internal error: <string>
```

これらのエラーは発生してはならないエラーです。発生した場合は、DOORS Analyst サポートにご連絡ください。

TSX0026: ポートに 2 つの in part または out part を含めることはできません

ツールを通常の方法で使用している場合、このエラーは発生しません。

TSX0047: タグ付き値はここでは使用できません

たとえば、クラス シンボル内などでは、プロパティ（タグ付き値）を編集できない場合があります。ステレオタイプ自体のみを追加できます。

プロパティの編集には、プロパティ エディタを使用することを推奨します。

TSC: セマンティック チェック

セマンティック チェックについて

セマンティック チェックでは、UML モデルが完全で、言語構成要素間の関係に意味があることをベリファイします。

モデル内に不完全な構成要素が含まれていると、セマンティック エラーが発生します。サポートされている構成要素を識別するには、UML 言語ガイドが役立ちます。

このフェーズで発生するエラーと警告には、TSC という接頭辞が付きます。

TSC0123: 再帰的な依存が、%n の定義で見つかりました <string>%s 経由)

これは循環依存性エラーです。2 つのクラスを同時に他のクラスのコンテナにすることはできないので、これは不正と見なされます。

このタイプのエラーの例を以下に示します。

例 1

```
class X {
    part Y y;
}

class Y {
    part X x;
}
```

TSC0134: C コードを生成する場合、遷移は、stop、nextstate または join action で終了する必要があります

分岐では、「else」などのすべての答えの可能性を含める必要があります。

TSC0092: 対応する 'virtual(仮想)' または 'redefined(再定義)' 操作が親シグニチャで見つかりませんでした (または存在しません)。

このエラーは要因としてさまざまな状況が考えられます。以下の例は、起こり得る状況を示しています。

汎化がないアクティブ クラスでの再定義操作の使用

例 2: 汎化のないクラス

```
active class P {
```

```
    redefined void Op() { }
}
```

アクティブ クラスの汎化で再定義操作を使用すると、このエラーが発生する場合があります。

例 3: 親クラスでの操作の不一致

```
active class P {
}
active class C :P {
     redefined void Op() { }
}
```

親クラスでの操作（Op）に異なるシグニチャがあると、以下のような状況になる場合があります。

例 4: 仮想性は「virtual」または「redefined」でなければならない

非仮想操作を再定義することはできません。

```
active class P {
     void Op () {   }
}
active class C :P {
    redefined void Op() { }
}
```

例 5: 異なる戻り型

```
active class P {
    virtual Integer Op () { return 1; }
}
active class C :P {
    redefined void Op() { }
}
```

例 6: 異なる仮パラメータのカウント

```
active class P {
    virtual void Op (Integer x) {  }
}
active class C :P {
    redefined void Op() { }
}
```

例 7: 異なるタイプの仮パラメータ

```
active class P {
```

```
    virtual void Op (Integer x) {  }
}
active class C :P {
    redefined void Op(Real x) { }
}
```

TSC0196: ファイナライズされた操作を再定義することはできません。

親クラス内の操作はファイナライズされていますが、子と同じシグニチャになっています。

例 8: ファイナライズされた操作

```
active class P {
    finalized void Op () { }
}
active class C :P {
    redefined void Op() { }
}
```

TSC0236: 操作 '<name>' はポート上で 'Realized' (実現化) として指定することはできません。

このチェックによって以下のケースが検出されます。

```
active class <class name>
{
    port <port name> in with <in_name>;
}
```

この場合の <in_name> は、同じ名前の操作にバインドされます。

例 9

```
active class a {
   void foo()  {}
   port p in with foo;
}
```

これはエラーとして報告されます。この問題を解決するには、アクティブ クラス a のインターフェイスで foo() を定義する必要があります。

TSC0237: 操作 '<name>' はポート上で 'Required' (要求) として指定することはできません。

このチェックによって以下のケースが検出されます。

```
active class <class name>
```

```
{
    port <port name> out with <out_name>;
}
```

この場合の <out_name> は、同じ名前の操作にバインドされます。

例 10

```
active class a {
   void foo()  {}
   port p out with foo;
}
```

これはエラーとして報告されます。この問題を解決するには、アクティブ クラス a のインターフェイスで foo() を定義する必要があります。

TSC2300: 式 'any (type)' には interface 型を定義できません

このタイプのエラーの例を以下に示します。

例 11

```
interface I {
}

active class X {
    Integer Op () {
        switch (any (I)) {
            case 5  :{ return 1; }
            default :{ return 0; }
        }
    }
}
```

TSC2302: データ型からの関連は、誘導可能リモート関連を終端にすることはできません。

データ型に属性を設定できないので、データ型からの関連付けがあると不正と見なされます。誘導可能性は常にデータ型に従っている必要があります。

ツールを通常の方法で使用している場合、このエラーは発生しません。

TSC2303: 関連の 1 つの終端のみを集約または合成にできます。

集約と合成は異なる種類の「part-of」構成要素なので、2 つのクラスを異なるクラスのコンテナにすることはできません。

例 12

この状況は 297 ページの図 2 のような状況で発生します。

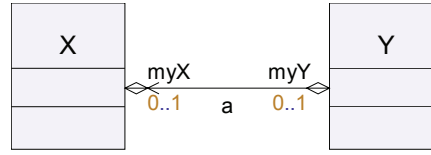

図 2: 循環参照のあるクラス

TSC2304: パートでない属性には、初期カウントを与えることはできません。

UML では、正規の属性に初期カウントを指定できません。パートのみに有効です。

このタイプのエラーの例を以下に示します。

例 13

```
class Z {
    Integer [1..*] a / 1;
}
```

TSC2305: パートにはデフォルト値を与えることはできません。

パートはアクティブ クラスのインスタンスなので、デフォルト値を指定できません。このタイプのエラーの例を以下に示します。

例 14

```
active class X {
    part Y a = 10;
}
```

TSC2306: 合成属性や関連の終端を、データ型により型指定することはできません。

UML でパートと見なされる合成属性にも、データ型のインスタンスを使用できません。

このタイプのエラーの例を以下に示します。

例 15

```
class X {
    part Integer d;
}
```

TSC2307: 合成属性にはこの属性を所有する型を (直接または間接的に) 指定できません。

これは循環依存性エラーです。クラスはそれ自体のコンテナにできないので、これは不正と見なされます。

このタイプのエラーの例を以下に示します。

例 16

```
class X {
    part X y;
}
```

TSC2308: 呼び出し式の 'via' は、ポートを参照する必要があります。

このタイプのエラーの例を以下に示します。

例 17

```
class Y {}
signal sig ();
active class X {
    port p out with sig;
    void Op () {
        output sig via Y;
    }
}
```

TSC0269: インターフェイス I とクラス Y の間の汎化は無効です。

このタイプのエラーの例を以下に示します。

例 18

```
class Y {
}
interface I :Y {
```

```
}
```

TSC2325: 継承の循環

このエラーは、シグニチャが直接または間接的にそれ自体に基づいている場合に発生します。

このタイプのエラーの例を以下に示します。

例 19

```
class X :Y {
}
class Y :X {
}
```

TSC4001: C コードを生成する場合、戻り値は代入式の左辺で処理する必要があります。

たとえば、操作を返す値からの戻り値は無視できません。そのような戻り値は、たとえば属性に保存する必要があります。

例 20

整数を返す以下のような操作 Op を考えます。

```
Op ():Integer
```

ここで Op を呼び出します。

```
...
Integer i;
...
i=Op(); // Correct way of calling Op
Op(); // Error is reported
...
```

このチェックが実行されるのは、意味チェッカーがいずれかの C コード ジェネレータとビルド タイプが関連するビルドのコンテキストで実行される場合のみです（モデル ベリファイヤ（Model Verifier）、C コード ジェネレータ、AgileC コード ジェネレータ）。

TNR: 名前解決

名前解決では、UML エンティティの名前を識別して、モデル内の正しい定義にバインドします。名前解決エラーは、モデル内の不一致によって発生します。たとえば、不明瞭になり確実には解決できないような名前の変更をエンティティ上で行うとこのエラーが発生します。

このフェーズで発生するエラーと警告には、TNR という接頭辞が付きます。

エラーの**主体**が UML エンティティではなくプロジェクト ファイル（`.ttp` ファイル）に関連している TNR エラーの場合は、必ず DOORS Analyst サポートに報告してください。

TNR0023: 要素の参照 <name> を見つけることができませんでした

名前バインドでは、名前を使用して現在の範囲内のエンティティを参照します。GUID バインドとは、エンティティがその固有な ID（GUID）によって参照されるという意味です。つまり、何らかの理由でエンティティが削除され、モデル内のいずれかの場所でその GUID によって参照されるとエラーが発生します。

問題を解決するには、正しい GUID と一緒にエンティティをロードして、参照を削除するか名前バインドが使用されるように参照を変更します。

UML へのインポートとエクスポート

「UML へのインポートとエクスポート」セクションの各章では、それぞれ固有のフォーマットをもつ、他のツール間で情報をやりとりする方法 について説明しています。

6

UML インポート

この章では、Telelogic DOORS Analyst 以外の UML ツールで作成された UML モデルとダイアグラムのインポートについて説明します。

動作原理

XMI

XMI - XML メタデータ交換は、異なる（個別の）ツール間での UML モデルの交換を可能にする XML に基づいた UML メタデータ表現の標準です。XMI DTD（XML Document Type Definitions）は、XMI ドキュメントの構文仕様を提供します。この仕様に従って汎用 XML ツールで XMI ドキュメントのコンポーズおよびバリデートを行うことができます。

UML メタ モデル クラスは、XMI DTD 内でその名前がクラス名である XML 要素によって表されます。要素定義によってクラスの属性が記述されます。クラスに関連する関連付けの終端の参照、明示的またはコンポジションの関連付けによってネストされたクラスなどが対象になります。

メタモデル クラスの属性は、DTD 内でその名前が属性名である XML 要素によって表されます。

メタモデル クラス間の関連付け（包含ありと包含なしの両方）は、関連付けの終端の役割を表す 2 つの XML 要素によって表されます。

XMI インポート

UML インポート実行時に、XMI 標準に準拠したファイルを読み込み、XMI ファイルの内容を解釈してから UML モデルを作成します。インポートが完了すると、インポートされた内容を可視化するためにプレゼンテーション要素（ダイアグラムとシンボル）が作成されます。また、インポートされた XMI ファイルにダイアグラムとシンボルの情報が含まれている場合は、そのような情報を使用して、インポート後の UML モデルの表示が保持されます。

ダイアグラム情報のない XMI ファイルは、インポートされますが、UML モデル要素のみが作成されます。

XMI インポート アドイン

XMI インポート機能は、**XMIImport** という名前のアドインによって提供されます。

XMI インポート アーキテクチャ

この機能のアーキテクチャの概要を、313 ページの図 1 に示します。XMI Reader モジュールが XMI 仕様のファイルを読み込みます。このモジュールは、それぞれのタグから情報を変換してその情報を UML API に渡します。

UML モデルの要素は、すべて UML API で作成されます。UML API のコア部分は、UML メタモデルの場合と同じクラス階層を持つ C++ クラス セットになっています。UML API は、動的に（タグごとに）UML モデルのスケルトンを作成する（XMI Reader と連携して）モジュール ビルダです。

このフェーズでは、一部の情報を UML モデルに追加できない場合があります。そのような情報は収集されて UML Resolver モジュールに渡されます。

U2 Resolver は、UML モデルのスケルトンに対して一連の変換を実行します。

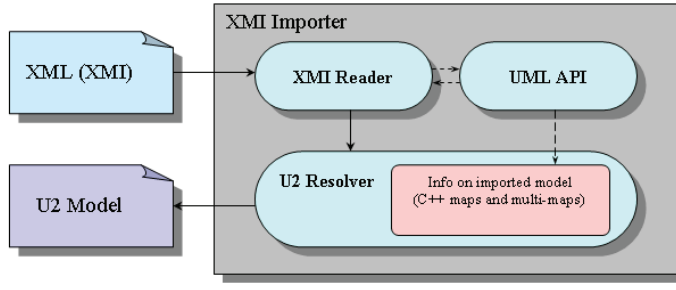

図 1: XMI インポート アーキテクチャ

例 1: UML Resolver

U2 Resolver に渡される情報が「列挙型」データ型のインポートの場合の例を示します。たとえば、Rational Rose は、「列挙」を <<enumeration>> によってステレオタイプ化されたクラスとしてエクスポートしますが、DOORS Analyst では「列挙」は `DataType` となります。適用されたステレオタイプに関する情報は、クラス インポート時に提供されません。したがって、このクラスは後から変換する必要があります。必須変換に関する情報は、ステレオタイプ インポート時に U2 Resolver に渡されます。

XMI ファイルのインポート

XMI インポートは、DOORS Analyst GUI から呼び出します。XMI インポートを開始するには、プロジェクトが含まれるワークスペースを開く必要があります。

- ［モデル ビュー］で**パッケージ**を選択します（詳細レイアウトを使用します。ワークスペース ウィンドウにビュー タブがあります）。
- インポート ウィザードを開きます（［ファイル］メニューの［インポート］コマンド）。
- ダイアログで［XMI のインポート］を選択して、［OK］をクリックします。
- 表示されるダイアログで、インポートする XMI ファイルを指定します。

2 番目のダイアログを閉じると、以下のような結果になります。

- パッケージ **ImportedXMIDefinitions** がモデル内に作成されます。
- ステレオタイプ **xmiImportSpecification** がパッケージに適用されます。

- インポートする XMI ファイルが、パッケージのステレオタイプ インスタンスに値として格納されます。
- インポート操作が実行されて、作成されたパッケージに結果が追加されます。

同じ設定を使用する XMI 仕様の再インポート

［モデル ビュー］で、**xmiImportSpecification** ステレオタイプが適用されているパッケージを選択します。

パッケージを右クリックして、ポップアップ メニューから［XMI のインポート］を選択します。

インポート操作が実行されて、結果がパッケージに追加されます。

設定を変更するには（インポートするファイルを選択）、［XMI のインポート］コマンドを実行する前に、パッケージのステレオタイプ インスタンスでプロパティを編集できます。

注記

［インポート ウィザード］ダイアログを使用すると、新しいパッケージが作成されます。ポップアップ メニューから［XMI のインポート］を再度使用すると、既存のパッケージが再利用されます。

サポートされる XMI と UML

言語とバージョンのサポート

以下の言語とバージョンが XMI インポートでサポートされています。

- XMI 1.0/1.1
- UML 1.4

XMI インポートでサポートされている UML 1.4 エンティティを、以下に示します。特に指定がない限り、エンティティの関係と属性もサポートされます。

基底／コア

- 関連（Association）
- 関連の終端（AssociationEnd）
- 属性（Attribute）
- クラス（Class）
- コメント（Comment）
- コンポーネント（Component）
- 制約（Constraint）
- データ型（DataType）
- 依存（Dependency）
- 要素常駐場所（ElementResidence）

- 列挙（Enumeration）
- 列挙リテラル（EnumerationLiteral）
- 汎化（Generalization）
- インターフェイス（Interface）
- メソッド（Method）
- 操作（Operation）
- パラメータ（Parameter）
- 許可（Permission）
- 構造特性（StructuralFeature）

基底／拡張

- ステレオタイプ（Stereotype）
- タグ付き値（TaggedValue）
- タグ定義（TagDefinition）

基底 / データ型

- ブール値（Boolean）
- 論理式（BooleanExpression）
- 式（Expression）
- 整数（Integer）
- 多重度（Multiplicity）
- 多重度範囲（MultiplicityRange）
- 名前（Name）
- プロシージャ式（ProcedureExpression）
- 文字列（String）
- 未解釈（Uninterpreted）

モデル管理

- モデル（Model）
- パッケージ（Package）
- サブシステム（Subsystem）

振る舞い要素／共通振る舞い

- アクション シーケンス（ActionSequence）
- 引数（Argument）
- 呼び出しアクション（CallAction）
- 生成アクション（CreateAction）
- 消滅アクション（DestroyAction）
- 例外（Exception）

- 戻りアクション（ReturnAction）
- 送信アクション（SendAction）
- シグナル（Signal）
- 終了アクション（TerminateAction）
- 未解釈アクション（UninterpretedAction）

振る舞い要素／コラボレーション

- 分類子ロール（ClassifierRole）
- コラボレーション（Cllaboration）
- 相互作用（Interaction）
- メッセージ（Message）

振る舞い要素／ユース ケース

- アクター（Actor）
- 拡張（Extend）
- 包含（Include）
- ユース ケース（UseCase）

振る舞い要素 / 状態機械

- 合成ステート（CompositeState）
- 呼び出しイベント（CallEvent）
- 終了状態（FinalState）
- ガード（Guard）
- 擬似ステート（Pseudostate）

```
Initial
Choice
Junction
DeepHistory
ShallowHistory
```

- シグナル イベント（SignalEvent）
- ステート（State）
- 単純ステート（SimpleState）
- 状態機械（StateMachine）

サポートされるダイアグラム タイプ

XMI ファイルに必須ダイアグラム情報が含まれている場合に、XMI インポートでは以下の UML ダイアグラム タイプをサポートします。

- クラス図
- コンポーネント図
- 配置図

- パッケージ図
- アクティビティ図
- シーケンス図
- ユース ケース図
- 状態機械図

保存されたレイアウトを使用するインポート

このカテゴリのダイアグラムは、XMI ファイルに図形的なレイアウトが存在するダイアグラムです。

- クラス図
- コンポーネント図
- 配置図
- パッケージ図
- アクティビティ図
- ユース ケース図
- シーケンス図
- 状態機械図

ネストされたステートのインポート

レイアウトは保持されますが、ネストされたステートには特殊な配慮を適用します。

- ステートがネストされているステートごとに、一連のダイアグラムが作成されます（ネスト レベルごとに 1 つ）。
- これらのダイアグラムのステートの位置は、可能な限り元と同じように保持されます。
- 開始シンボルと戻りシンボルは、必要に応じて新しい各ダイアグラム上に作成されます。これらのシンボルの位置は、可能な限り高位ネスト レベル上の対応するシンボルの位置と同じに保持されます。
- 新しいエントリと終了の接続ポイントは、必要に応じて作成されます。
- 大量のテキストが含まれている遷移イベントとアクションは重複する場合があります。

UML 1.x ツールからのインポート

通常の場合、XMI インポート ツールは、サポートされる XMI バージョンに準拠している以下の UML 1.x ツールからの XMI ファイルをサポートします。

- Rational Rose/Unisys (JCR.2 v.1.3.x)
- DOORS Analyst UML スイート
- Borland Together
- IBM XMI ツールキット

Rhapsody

Rhapsody は XMI をエクスポートしますが、ダイアグラム情報は含まれません。Rhapsody からエクスポートされた XMI ファイル内の情報を使用して、モデル要素が作成されます。その結果、UML 構造がワークスペース ウィンドウに表示されます（ダイアグラムは含まれません）。

Rational Rose

- Unisys の拡張機能を使用する Rational Rose は、XMI をダイアグラム情報と一緒にエクスポートします。この情報は、ダイアグラム作成時の XMI インポート時に使用されます（ダイアグラムがサポートされるダイアグラム タイプのいずれかであることが前提になります）。
- ダイアグラム レイアウトは、クラス図、ユース ケース図、シーケンス図用に保持されます。
- Rational Rose 名はサポートされます。

DOORS リンクの保持

Rational Rose からの XMI のインポートの間に DOORS のリンクを保持できます。

- UML モデルをエクスポートします。［Generated UUIDs］チェックボタンが選択されていることを確認します。
- 生成された XMI を DOORS Analyst にインポートします。
- 既存の DOORS インテグレーション コマンドを使用して、DOORS Analyst から DOORS へ新しい UML をエクスポートします。
- DOORS 内で DOORS Analyst 代理モジュールを開き、［Import Links from Rational Rose］メニューを選択して指示に従います。

この操作が完了すると、DOORS 内の代理モジュールとの間のすべてのリンク（DOORS Rose Link インテグレーションにて作成された）は、DOORS Analyst 代理モジュールのためにコピーされます。

DOORS Analyst UML スイート

- DOORS Analyst UML スイートが Unisys の拡張機能を使用している場合、XMI をダイアグラム情報と一緒にエクスポートします。この情報は、ダイアグラム作成時の XMI インポートに使用されます（ダイアグラムがサポートされるダイアグラム タイプのいずれかであることが前提になります）。
- ダイアグラム レイアウトは、クラス図、ユース ケース図、シーケンス図用に保持されます。

参照

言語とバージョンのサポート

制限事項

この章では XMI ／ UML サポートのレベルについて説明していますが、以下のセクションではさらにその他の既知の制限について説明します。

タイプと変数の定義

- ローカル データ型定義は状態機械図に表示されません。
- ローカル変数定義は状態機械図に表示されません。

不完全なモデル

XMI 仕様をインポートするには、完全で、セマンティックの正しい UML モデルになっている必要があります。通常の場合、不完全または不正な仕様は DOORS Analyst にインポートできませんが、そのような仕様を完全な仕様としてインポートするか、インポート時に一部の情報が失われる場合があります。

例 2: 不完全なモデルのインポート

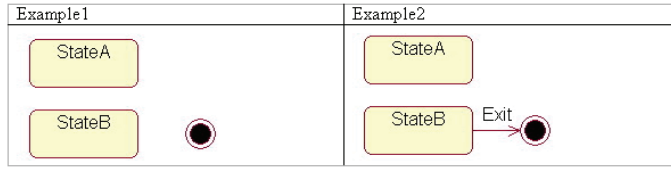

図 2: 不完全なモデル

例 1 では、FinalState がインポートされません。このステートが ReturnAction に変換されるためです。このアクションは FinalState に入る遷移によって所有される必要があります。そのような遷移はこの例では存在しないため、FinalState はインポートされません。

例 2 では、すべてのダイアグラム要素がインポートされますが、このダイアグラムも不完全です（このダイアグラムには InitialState がありません)。

サポートされないクラス

UML 構成要素には 、XMI インポート実行中に処理されないものがあります。

以下の構成要素の場合は、エラー メッセージ TUI0004（サポートされていないクラス）が出力されます。

基底：コア

- アーティファクト（Artifact）

- 関連（Association）（ユース ケース間）
- バインディング（Binding）
- フロー（Flow）
- 汎化（Generalization）（アクター間）

振る舞い要素：共通振る舞い

- 属性リンク（AttributeLink）
- コンポーネント インスタンス（ComponentInstance）
- データ値（DataValue）
- インスタンス（Instance）
- リンク（Link）
- リンク終端（LinkEnd）
- ノード インスタンス（NodeInstance）
- オブジェクト（Object）
- 受信（Reception）
- 刺激（Stimulous）
- サブシステム インスタンス（SubsystemInstance）

振る舞い要素：アクティビティ グラフ（ActivityGraphs）

- ステート内の分類子（ClassifierInState）
- オブジェクト フロー状態（ObjectFlowState）
- 擬似ステート（Pseudostate）（浅い履歴と詳細な履歴）（Shallow history と Deep history）

振る舞い要素：コラボレーション

- 関連終端ロール（AssociationEndRole）
- 関連ロール（AssociationRole）
- コラボレーション インスタンス セット（CollaborationInstanceSet）
- 相互作用インスタンス セット（InteractionInstanceSet）

振る舞い要素：状態機械

- 変更イベント（ChangeEvent）
- スタブ ステート（StubState）
- タイム イベント（TimeEvent）

振る舞い要素：ユース ケース

- ユース ケース インスタンス（UseCaseInstance）

サポートされない属性

以下の属性の場合は、サポートされていない属性のエラー メッセージ（TUI0006）が出力されます。

基底：コア

- 関連終端（AssociationEnd）
  - 仕様（Specification）
- 属性
  - 関連終端（AssociationEnd）
- 振る舞い特性（BehavioralFeature）
  - 発生シグナル（RaisedSignal）
- コンポーネント（Component）
  - デプロイメント（Deployment）
- 制約（Constraint）
  - 制約付きステレオタイプ（ConstrainedStereotype）
- 特性（Feature）
  - 所有者（Owner）
- メソッド（Method）
  - 本体（Body）
  - 所有者スコープ（OwnerScope）
- モデル要素（ModelElement）
  - プレゼンテーション（Presentation）
  - テンプレート（Template）
- 操作（Operation）
  - 並列性（Concurrency）
  - オカレンス（Occurence）
  - 仕様（Specification）

基底：データ型

- 式（Expression）
  - 言語（Language）

基底：拡張

- ステレオタイプ（Stereotype）
  - アイコン（Icon）
  - ステレオタイプ制約（StereotypeConstraint）

振る舞い要素：コラボレーション

- コラボレーション（Collaboration）
  - 記述分類子（RepresentedClassifier
  - 記述操作（RepresentedOperation）
- 相互作用（Interaction）
  - コンテキスト（Context）
- メッセージ（Message）
  - アクティベータ（Activator）

振る舞い要素：状態機械

- 合成ステート（CompositeState）
  - 並列プロパティ（Concurrent property）

振る舞い要素：ユース ケース

- アクター（Actor）
  - 抽象プロパティ（Abstract property）
- ユース ケース（Use Case）
  - 拡張ポイント（ExtensionPoint）

モデル管理

- サブシステム（Subsystem）
  - インスタンス化可能プロパティ（Instantiable porperty）

サポートされないコンポジション

以下の構造の場合は、サポートされていないコンポジションのエラー メッセージ（TUI0008）が出力されます。

基底：コア

- 関連終端（AssociationEnd）
  - 修飾子（Qualifier）
- コンポーネント（Component）
  - 実装（Implementation）

振る舞い要素：アクティビティ グラフ（ActivityGraphs）

- ステート
  - 内部遷移（InternalTransitions）（アクティビティのアクション）
  - ステート（State）（状態機械に基づく）
  - 擬似ステート：履歴（Pseudostate: History）（浅い履歴と詳細な履歴）（Shallow history と Deep History）

振る舞い要素：コラボレーション

- コラボレーション
  - 制約要素（ConstrainingElement）

振る舞い要素：状態機械

- 状態機械（StateMachine）
  - 同期ステート（SynchState）（同期バー）
- ステート
  - 内部遷移（InternalTransitions）（ステートのアクション）
  - 擬似ステート：ジャンクション（Psuedostate:Junction）

コラボレーション図はサポートされていません。

エクスポートの制限事項

Rational Rose は、不完全なエクスポートを実行する場合があります。その結果、インポート後に一部の情報が失われることがあります。Rational Rose エクスポータ（Unisys 1.3.6）の既知の問題（エクスポートされない機能）を以下に示します。

クラス図

- クラス
  - タイプ（ParameterizedClass、ClassUtility、InstantiatedClass など）
  - 多重度（Multiplicity）
  - スペース（Space）
  - 並列性（Concurrency）
  - 形式（Format）（表示可能性）
- 属性（Attribute）
  - 包含（Containment）

- 操作（Operation）
  - プロトコル（Protocol）
  - 修飾（Qualification）
  - サイズ（Size）
  - 時間（Time）
- バイナリ関連（Binary Association）
  - 制約（Constraints）
  - 包含（Containment）
  - 派生（Derived）
  - フレンド（Friend）
  - リンク要素（LinkElement）
  - 名前ディレクション（Name Direction）
- 継承（Inheritance）
  - 文書化（Documentation）
  - 仮想継承（Virtual inheritance）
  - フレンドシップ必須（Friendship Required）
- 実現化（Realization）
  - 文書化（Documentation）
- 依存 / インスタンス化（Dependency/Instantiates）
  - 多重度（こちらから）（Multiplicity from）
  - 多重度（こちらへ）（Multiplicity to）
  - フレンドシップ必須（Friendship Required）

ステート図

- 遷移（Transition）
  - ステレオタイプ（Stereotype）
  - 文書化（Documentation）

シーケンス図

- メッセージ（Message）
  - 頻度（Frequency）（周期的、非周期的）
- 破棄マーカ（Destruction marker）

ユース ケース図

- アクター（Actor）
  - タイプ（Type）
  - 多重度（Multiplicity）
- ユース ケース（Use Case）
  - ステレオタイプ（Stereotype）
  - ランク（Rank）
- バイナリ関連（Binary Association）
  - 派生（Derived）
  - リンク要素（Link Element）
  - 名前ディレクション（Name Direction）
  - 制約（Constraints）
  - フレンド（Friend）
  - 包含（Containment）
- 依存（Dependency）

パッケージ図

- 依存（Dependency）
  - 文書化（Documentation）

コンポーネント図

- パッケージ（Package）
  - グローバル（Global）
- コンポーネント（Component）
  - 宣言（Declarations）

配置図

- プロセッサ（Processor）
  - スケジューリング（Scheduling）
- プロセス（Process）
  - 優先順位（Priority）
- デバイス（Device）
  - ステレオタイプ（Stereotype）
- 接続（Connection）

アクティビティ図

- スイムレーン（Swimlane）
  - 文書化（Documentation）
- オブジェクト（Object）
- オブジェクト フロー（Object Flow）

エラー メッセージ

概要

XMI インポート中のエラーメッセージは 出力ウィンドウに表示されます。

XMI インポートのエラーメッセージ

コード	テキスト	コメント
TUI0004	属性 '<name>'(<name>)(クラス '<name>') はサポートされていません	XMI 仕様に XMI 標準で指定されていない属性が含まれているか、属性を Tau 内の現在のクラスに適用できないときにエラーが発生します。たとえば、クラス「アクター」の属性「isAbstract」は、Tau に適用できません。
TUI0006	合成 '<name>'(クラス '<name>' からクラス '<name>') はサポートされていません	このエラー メッセージは、クラス間の合成がサポートされていないときに出力されます。たとえば、「修飾子」合成は Tau でサポートされていません。
TUI0008	クラス <name> はサポートされていません	インポートされた XMI 仕様に、たとえば、サポートされていないクラス「インスタンス」が含まれているとエラーが発生します。
TUI0009	グラフィック要素 '<name>'(クラス '<name>') は描かれていません	このエラー メッセージは、PresentationElement の対応する ModelElement が見つからないときに出力されます。たとえば、対応する ModelElement はサポートされていません。
TUI0010	ダイアグラム形式 '<name>'(値 '<name>') はサポートされていません	プレゼンテーション要素が Tau によってサポートされていないときにエラーが発生します。たとえば、ステレオタイプの PresentationElement はサポートされていません。
TUI0016	ファイル <name> を開けませんでした	XMI Importer に渡されたファイルは開けません。
TUI0017	XMI ファイルの分析中に分析エラーが検出されました	このエラー メッセージは、XML パーサで情報を XMI 仕様から読み込めないときに出力されます。たとえば、XML パーサは終了タグを発見できません。
TUI0022	内部エラーです	内部エラーはインポート時に発生します。

7

UML1.x XMI エクスポート

本章では、UML1.x を使用するツールへの XMI 形式のモデル データのエクスポートを DOORS Analyst がどのようにサポートするかについて説明します。

XMI のエクスポート

操作の原理

UML エクスポータは、XMI 標準に準拠するファイル形式を生成します。エクスポート時、モデル要素とプレゼンテーション要素の両方がファイルに書き出されます。Unisys XML プラグインに基づいて UML モデルの外観を保持するため、ダイアグラムとシンボルのレイアウト情報が含まれます。

XMI エクスポート アドイン

XMI エクスポート機能は、**XMIExport** という名前のアドインで提供されます。

XMI ファイルへのエクスポート

XMI エクスポータは、DOORS Analyst GUI から呼び出します。

- ［XMI エクスポート］ウィンドウを開きます（［ツール］メニューから［モデルを XMI にエクスポート］コマンドを選択します）。
- 表示されるダイアログで、エクスポート先の XMI ファイルを指定します。

ワークスペースに複数のプロジェクトが存在する場合、XMI エクスポートは選択したプロジェクトに対して行われます。プロジェクトを選択しなかった場合、また複数のプロジェクトを選択した場合、メニュー項目がグレー表示となります。

サポートされる XMI とツールのバージョン

XMI エクスポータは以下のバージョンをサポートします。

- XMI 1.1

XMI エクスポータは、以下のターゲット ツール環境でテストされています。

- Rational Rose Enterprise Edition 2003
- Rose XML Tools (UniSys XML plug-in) 1.3.6 for Rational Rose

サポートされる UML エンティティ

以下の表に、XMI エクスポータがサポートする DOORS Analyst UML エンティティの一覧を示します。

- **UML エンティティ**
  DOORS Analyst 内の UML エンティティ。
- **エクスポート**
  DOORS Analyst からエクスポートして Rose にインポートした場合、Rose 内でのインポート結果のエンティティ。

- **ラウンドトリップ**
  XMI ラウンドトリップを実行した場合、DOORS Analyst 内でのインポート結果のエンティティ

このリストにない他のエンティティはエクスポートされません。

UML ダイアグラム	エクスポート	ラウンドトリップ
アクティビティ図	同じ	同じ
クラス図	同じ	同じ
コンポーネント図	クラス図	クラス図
配置図	クラス図	クラス図
パッケージ図	クラス図	クラス図
シーケンス図	同じ	同じ
状態機械図	同じ	同じ
テキスト図	ノート付きクラス図	ノート付きクラス図
ユース ケース図	クラス図	クラス図

全般	エクスポート	ラウンドトリップ
コメント シンボル	ノート	同じ
注釈ライン	アンカー	同じ
テキスト シンボル	ノート	コメント シンボル
<Any>		
コメント	ドキュメント	なし
ステレオタイプ	同じ	同じ
リンク	ファイル	なし
色	同じ	同じ
フォント	同じ	同じ

アクティビティ図	エクスポート	ラウンドトリップ
アクティビティ シンボル	同じ	同じ
• 名前	同じ	同じ
アクション	「Entry」アクション	なし

アクティビティ図	エクスポート	ラウンドトリップ
アクティビティ	<<activity>> でステレオタイプ化されたクラス	<<activity>> でステレオタイプ化されたクラス
開始ノード	同じ	同じ
アクティビティ終了	終了ステート	アクティビティ終了
フロー終了	終了ステート	アクティビティ終了
アクティビティ ライン	遷移	同じ
テキスト	遷移ラベル	同じ
フォーク／ジョイン	同期	同じ
分岐	同じ	同じ
• 名前	同じ	同じ
SendSignalSymbol	「Do/send」アクションを持つ名称未設定アクティビティ	アクションのない自動名前変更済みアクティビティ
AcceptEventSymbol	「Do/receive」アクションを持つ名称未設定アクティビティ	アクションのない自動名前変更済みアクティビティ
AcceptTimeEventSymbol	「Do/receive」アクションを持つ名称未設定アクティビティ	アクションのない自動名前変更済みアクティビティ

クラス図	エクスポート	ラウンドトリップ
クラス	同じ	同じ
• 名前	同じ	同じ
• 抽象	同じ	同じ
• テンプレート パラメータ	仮引数	同じ
• 可視性	エクスポート コントロール	同じ
クラス属性	同じ	同じ
• 名前	同じ	同じ
• タイプ	同じ	同じ

クラス図	エクスポート	ラウンドトリップ
• 可視性	エクスポート コントロール	同じ
• デフォルト値	初期値	同じ
• 派生	同じ	同じ
クラス操作	同じ	同じ
• 名前	同じ	同じ
• 戻り型	同じ	同じ
• 可視性	エクスポート コントロール	同じ
• 指定された例外	例外	同じ
クラス操作パラメータ	同じ	同じ
• 名前	同じ	同じ
• タイプ	同じ	同じ
• デフォルト値	同じ	同じ
要求インターフェイス	インターフェイス	<<interface>> でステレオタイプ化されたクラス
実現化インターフェイス	インターフェイス	<<interface>> でステレオタイプ化されたクラス
インターフェイス	同じ	<<interface>> でステレオタイプ化されたクラス
タイマー	<<timer>> でステレオタイプ化されたクラス	<<timer>> でステレオタイプ化されたクラス
シグナル	<<signal>> でステレオタイプ化されたクラス	同じ
ステレオタイプ	<<stereotype>> でステレオタイプ化されたクラス	<<stereotype>> でステレオタイプ化されたクラス
操作	<<operation>> でステレオタイプ化されたクラス	<<operation>> でステレオタイプ化されたクラス
状態機械	<<statemachine>> でステレオタイプ化されたクラス	<<statemachine>> でステレオタイプ化されたクラス
基本／列挙	<<primitive>>/<<enumeration>> でステレオタイプ化されたクラス	<<primitive>>/DataType でステレオタイプ化されたクラス

クラス図	エクスポート	ラウンドトリップ
アーティファクト	<<artifact>> でステレオタイプ化されたクラス	<<artifact>> でステレオタイプ化されたクラス
コラボレーション	<<collaboration>> でステレオタイプ化されたクラス	<<collaboration>> でステレオタイプ化されたクラス
選択	<<choice>> でステレオタイプ化されたクラス	<<choice>> でステレオタイプ化されたクラス
関連ライン	同じ	同じ
• 名前	同じ	同じ
関連ロール	同じ	同じ
• 名前	同じ	同じ
• 可視性	エクスポート コントロール	同じ
制約	同じ	同じ
多重度	同じ	同じ
集約	集約、包含	同じ
所有者スコープ	静的	なし
汎化／実現化ライン	同じ	同じ
依存ライン	同じ	同じ
拡張ライン	<<extend>> でステレオタイプ化された依存	<<extend>> でステレオタイプ化された依存

コンポーネント図	エクスポート	ラウンドトリップ
コンポーネント シンボル	<<component>> でステレオタイプ化されたクラス	同じ

配置図	エクスポート	ラウンドトリップ
DeploymentSpecificationSymbol	<<deploymentSpecification>> でステレオタイプ化されたクラス	<<deploymentSpecification>> でステレオタイプ化されたクラス
ExecutionEnvironmentSymbol	<<executionEnvironment>> でステレオタイプ化されたクラス	<<executionEnvironment>> でステレオタイプ化されたクラス
NodeSymbol	<<node>> でステレオタイプ化されたクラス	<<node>> でステレオタイプ化されたクラス

パッケージ図	エクスポート	ラウンドトリップ
パッケージ	同じ	同じ
• 名前	同じ	同じ
依存ライン	同じ	同じ

シーケンス図	エクスポート	ラウンドトリップ
ライフライン	同じ	同じ
• 名前	同じ	同じ
• タイプ	クラス	同じ
メッセージ	単純なメッセージ	同じ
• 名前	同じ	同じ
メソッド呼び出し	プロシージャ呼び出しメッセージ	メッセージ
• 名前	同じ	同じ
メソッド応答	リターン メッセージ	同じ
• 名前	同じ	同じ
タイムアウト	タイムアウト メッセージ	メッセージ
• 名前	同じ	同じ
生成ライン	「:{Create}」という接尾辞の付いた名前のメッセージ	「:{Create}」という接尾辞の付いた名前のメッセージ
相互作用	<<interaction>> でステレオタイプ化されたクラス	<<interaction>> でステレオタイプ化されたクラス

状態機械図	エクスポート	ラウンドトリップ
ステート	同じ	同じ
• 名前	同じ	同じ
マルチ ステート（ステート リストまたはアスタリスク ステートを持つステート）	元のステート テキストに設定されたステート名を持つステート	元のステート テキストに設定されたステート名を持つステート
遷移ライン	同じ	同じ
ラベル	同じ	同じ
分岐	同じ	同じ
分岐質問	名前	同じ
分岐回答シンボル	遷移ガード条件	遷移ガード条件
開始	同じ	同じ
停止	終了ステート	リターン
リターン	終了ステート	同じ
フロー ライン	遷移	遷移
シグナル受信	遷移イベント	遷移イベント
ガード シンボル	遷移ガード条件	遷移ガード条件
アクション シンボル	遷移アクション	なし
シグナル送信	遷移送信イベント	なし

ユース ケース図	エクスポート	ラウンドトリップ
アクター	同じ	同じ
• 名前	同じ	同じ
• 可視性	エクスポート コントロール	同じ
ユース ケース	同じ	同じ
• 名前	同じ	同じ
パフォーマンス ライン	<<performance>> でステレオタイプ化された関連	同じ

ユース ケース図	エクスポート	ラウンドトリップ
依存ライン	同じ	なし
• 名前	同じ	なし
汎化ライン	同じ	同じ

モデルの階層

Rational Rose の包含階層は、338 ページの図 1 に示すような構造になります。

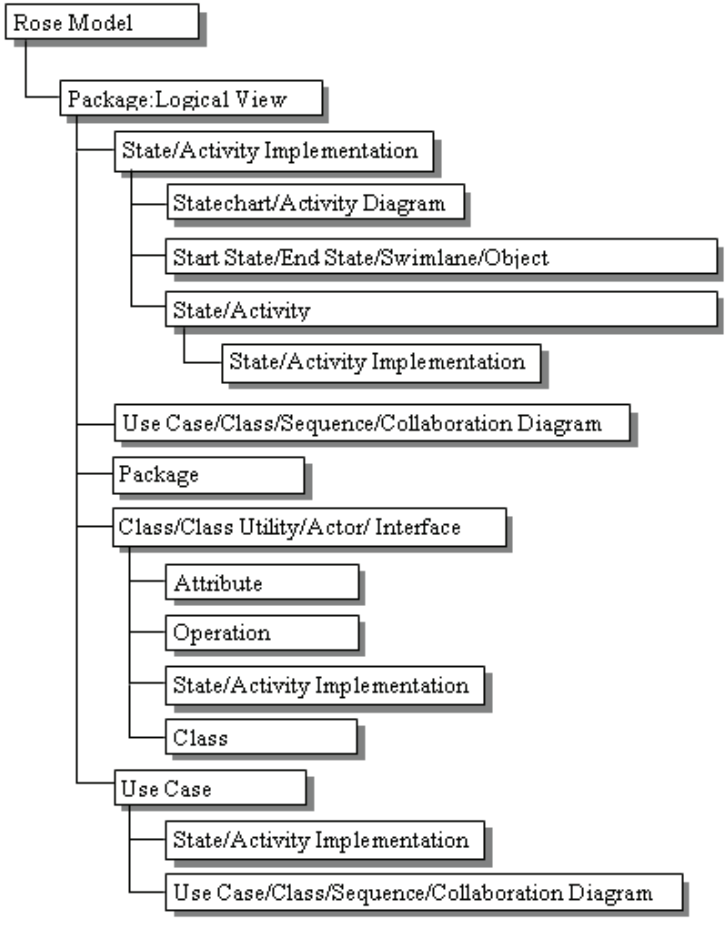

図 1: 包含階層

Rational Rose のビューはパッケージとして定義されます。Logical View は、ハードコードされた定義済みパッケージです。Rational Rose XMI モジュールがモデル要素とダイアグラムをインポートする際、デフォルトのインポート先となります。

State/Activity Implementations は状態機械仕様を表し、適用する要素の直下に置かれます。Packages とクラス Classes を（Classes、Class Utilities、Actors、Interfaces を介して）ネストできるので、階層は限りなく深くできます。どの要素の下にも File と URL を置くことができます。

一般的なルールとして、サポートされない XMI ファイルに包含すると、XMI インポート時に消失します。

モデル変換

モデル情報を可能な限り保持するため、モデルの変換が行われることがあります。

以下の表には次の内容を示します。

- DOORS Analyst エンティティ
- エクスポート後の XMI 内においてエンティティを移動する理由の説明
- DOORS Analyst エンティティに包含されていて上位階層に移動されるエンティティ

DOORS Analyst	説明	移動されるエンティティ
インターナル	同等な機能はありません。	Class diagram, Package diagram, Text diagram, UseCase diagram, Activity, Actor, Artifact, Association, Attribute, Choice, Class, Collaboration, DataType, Interaction, Interface, Operation, Signal, StateMachine, Stereotype, Timer, UseCase
状態機械実装	可能なエンティティのタイプに関して、このレベルは大きく制限されます。	Activity, Actor, Artifact, Association, Attribute, Choice, Class, Collaboration, DataType, Interface, Operation, Signal, Stereotype, Timer, UseCase, Class diagram, Package diagram, Text diagram, Use Case diagram
アクティビティ実装	可能なエンティティのタイプに関して、このレベルは大きく制限されます。	Actor, Artifact, Choice, Class, Collaboration, DataType, Interface, Signal, Stereotype, Timer, Use Case diagram
相互作用実装	同等な機能はありません。	Activity, Actor, Artifact, Attribute, Choice, Class, Collaboration, DataType, Interface, Operation, Signal, StateMachine, Stereotype, Timer, UseCase, Sequence diagram, UseCase diagram
ネストされたクラス	ネストされたクラスの下の状態機械図はインポートされません。	State machine diagram

DOORS Analyst	説明	移動されるエンティティ
クラス	クラスの下のインターフェイスはインポートされません。	`Interface`
属性	何かを包含することはできません。	`Artifact, Choice, Class, Collaboration, DataType, Interface, Stereotype`
選択	クラスに変換されます。	`UseCase`

Rational Rose への XMI エクスポートの制限事項

DOORS Analyst がエクスポートする XMI データについて、Rational Rose XMI Import では多くの制限事項があります。既知の問題について以下の表に示します。

一般的な機能	説明
可視性オプション	この設定は XMI を通しては伝達できません。DOORS Analyst で設定される可視性オプションと同じものがない場合は、ダイアグラム要素が重なることがあります。例として、クラス属性、操作、および操作シグニチャなどがあります。DOORS Analyst で無効にしていた可視性オプションを XMI データのインポート時に有効にした場合、インポート後のクラス シンボルのサイズが異なるため、シンボルの重なりが生じます。
ダイアグラム タイプ	ユース ケース図は、クラス図として Logical View にインポートされます。
ライン	インポートによってラインの頂点は消失します。
	ラインの色はインポートされません。
要素	同じダイアグラム内の 2 つ以上のシンボルのインスタンス（クラスなど）はインポートできません。
ノート	サイズはインポートされません。
	ノートはそのアンカーごとに 1 回複製されます。

アクティビティ図	説明
アクティビティ	アクティビティにステレオタイプとその下のサブアクティビティの両方がある場合、正しくインポートまたは表示されません。
	塗りつぶしの色、フォント、フォント サイズはインポートされません。
分岐	塗りつぶしの色、フォント、フォント サイズはインポートされません。
オブジェクト	インポートされません。

クラス図	説明
クラス	多重度はインポートされません。
	インターフェイスの下のクラスはインポートされません。
	ネストされたクラスの属性と操作はインポートされません。
クラス属性	Static(静的属性) はインポートされません。
インターフェイス	属性はインポートされません。
パッケージ	フォント、フォント サイズ、塗りつぶしの色はインポートされません。
関連	派生はインポートされません。
	制約はインポートされません。

パッケージ図	説明
パッケージ	塗りつぶしの色、フォント、フォント サイズはインポートされません。

シーケンス図	説明
ライフライン	関連付けられているテキストが長い場合、水平方向の間隔がエクスポート後の XMI で正しく表示されないことがあります。
	塗りつぶしの色、フォント、フォント サイズはインポートされません。

シーケンス図	説明
メッセージ	インポート時にメッセージ間の垂直方向のスペースが追加されます。
	同じ Y 座標を持つメッセージは、異なるレベルのライフラインに入れられます。
	線の色、フォント、フォント サイズはインポートされません。
破棄マーカー	インポートされません。
ノート	インポート時にメッセージのアンカーが接続されません。

状態機械図	説明
状態	同じダイアグラムにステートが複数回存在する場合、1 つのシンボルのみイポートされます。
	塗りつぶしの色、フォント、フォント サイズはインポートされません。
分岐	塗りつぶしの色、フォント、フォント サイズはインポートされません。
遷移ライン	線の色、フォント、フォント サイズはインポートされません。

ユース ケース図	説明
アクター	サイズはインポートされません。
	多重度はインポートされません。
ユース ケース	ステレオタイプはインポートされません。
	サイズはインポートされません。
依存	ユース ケース間に描画されている場合はインポートされません。

エラー メッセージと警告メッセージ

エラー メッセージと警告メッセージが出力ウィンドウの **XMIExport** というタブに表示されます。これらのメッセージはすべてナビゲート可能です。

XMI で表現できない UML エンティティについては、エラー メッセージが生成されます。

UML エンティティが変換された場合、または包含階層で移動された場合、警告メッセージが表示されます。これは、Rational Rose がこのような構造を処理できないことによります。

全タイプ共通のリファレンスガイド

このセクションの各章では、DOORS Analyst プロジェクトの全タイプに共通の機能のリファレンスを提供します。

8

印刷

この章では、ダイアグラムの各種印刷方法と印刷設定の変更方法について説明します。

ダイアグラムの印刷

ダイアグラムを印刷する複数の方法があります。以下の場所から 1 つのダイアグラムを印刷できます。

- ダイアグラム自体
- モデル ビュー
- 印刷マネージャ
- ダイアグラムのプレビュー ウィンドウ

以下の場所から複数のダイアグラムを印刷できます。

- モデル ビュー
- 印刷マネージャ

注記

アイコンイメージの背景を白または透明にすると、印刷時に背景が黒になる場合があります。この問題は Windows のポストスクリプト ドライバ PostScript 言語レベル 2 に起因しています。PostScript 言語レベル 1 に変更すると、問題を解決できます。カラーの背景またはフレームを使用して、この問題を解決することもできます。

印刷設定

印刷の設定を変更するには

1. ［ファイル］メニューから［印刷設定］を選択します。
2. ［プリンタの設定］ダイアログで、プリンタと用紙サイズなどの選択されたプリンタに使用できるプロパティを選択します。用紙サイズと印刷の向きは、エディタでデフォルトのダイアグラム サイズを決定する際に使用されます。
3. ［OK］をクリックします。
1. ファイルを印刷します。

ファイルを印刷するには

1. 印刷するファイルを開いて、カーソルをテキスト内の任意の場所に合わせます。
2. ［ファイル］メニューから［印刷］をクリックするか、ツールバーの印刷アイコンをクリックします。
3. ［印刷］ダイアログで、必要に応じて設定を変更します。
4. ［OK］をクリックします。

印刷するダイアグラムの選択

モデル内のすべてのダイアグラムが［モデル ビュー］に表示されます。印刷マネージャを使用して、印刷するダイアグラムを選択できます。印刷マネージャを開くには、［ファイル］メニューから［印刷マネージャ］をクリックします。

［モデル ビュー］でアクティブになっているコンテナ内のダイアグラムが印刷マネージャにリストされます。［モデル ビュー］でコンテナを変更する場合は、［選択部分をトラック］ボタンを使用します。ボタンを押さないと、［印刷マネージャ］に含まれている内容が最初の選択に従ってロックされます。

図 1: 選択されていない状態の［選択部分のトラック］ボタン

［フィルタ］領域でダイアグラム タイプのチェック ボックスを選択または選択解除して、どのタイプのダイアグラムを印刷するかを決定することもできます。

印刷するページ数を計算するには、［印刷マネージャ］で［ページ］をクリックします。

ダイアグラムのプレビュー

ダイアグラムのプレビューを表示するには

1. ［モデル ビュー］でダイアグラムを選択します。
2. ［ファイル］メニューから［印刷 プレビュー］を選択します。ダイアグラムのプレビューが表示されます。
   - ［次のページ］ボタンと［前のページ］ボタンを使用して、他のダイアグラムに移動できます。

1 つのダイアグラムの印刷

ダイアグラム自体から 1 つのダイアグラムを印刷するには

1. ダイアグラムを開きます。
2. ［ファイル］メニューから［印刷］を選択します。標準の［印刷］ダイアログが表示されます。

［モデル ビュー］から 1 つのダイアグラムを印刷するには

1. ［モデル ビュー］でダイアグラムを選択します。
2. ダイアグラムを右クリックして、［印刷］を選択します。標準の［印刷］ダイアログが表示されます。

［印刷マネージャ］から 1 つのダイアグラムを印刷するには

1. ［モデル ビュー］でダイアグラムを選択します。ダイアグラム アイコンが［選択］領域に表示されます。
2. ［印刷ビュー］ボタンをクリックします。標準の［印刷］ダイアログが表示されます。

プレビュー ウィンドウから 1 つのダイアグラムを印刷するには

- ［印刷］を選択します。標準の［印刷］ダイアログが表示されます。

複数のダイアグラムの印刷

［モデル ビュー］から複数のダイアグラムを印刷するには

1. ［モデル ビュー］で複数のダイアグラムを選択します。
2. ［ファイル］メニューから［印刷マネージャ］を選択します。［印刷マネージャ］ウィンドウが表示されます。
3. ［印刷ビュー］ボタンをクリックするか、ファイル］メニューから［印刷プレビュー］を選択して、［印刷］を選択します。標準の［印刷］ダイアログが表示されます。

［印刷マネージャ］ウィンドウと［フィルタ］機能を使用すると、同じタイプのダイアグラムを同時に印刷できます。

［印刷マネージャ］から複数のダイアグラムを印刷するには

1. ［ファイル］メニューから［印刷マネージャ］を選択します。［印刷マネージャ］ウィンドウが表示されます。
2. ［モデル ビュー］で、印刷するダイアグラムを選択します。選択したダイアグラムタイプのダイアグラムとページ番号が［選択］領域に表示されます。
3. ［印刷ビュー］ボタンをクリックするか、ファイル］メニューから［印刷プレビュー］を選択して、［印刷］を選択します。標準の［印刷］ダイアログが表示されます。

9

各国語対応サポート

このセクションでは、DOORS Analyst/Developer および Architect の各国語対応サポートについて説明します。このドキュメントの主要テーマは、中国語、日本語、韓国語（CJK）の言語処理です。

サポートされている環境

このセクションでは、システム環境の各国語対応サポートに関する特定の情報を提供します。ここで説明されていない情報は、すべての言語に共通しています。全般情報については、インストール ガイドを参照してください。

サポートされているプラットフォーム

DOORS Analyst の各国語対応サポートは、Windows 2000 と XP で有効です。Windows のローカル バージョンを使用して、ローカル言語用のロケールを設定することを前提にします。

構成管理

DOORS Analyst は、CJK サポートの各構成管理ツールの制限外の CJK 環境をサポートしていません。

IME（Input Method Editor）

Windows にバンドルされているデフォルトの IME をサポートしています。サポートされている IME を使用して、ローカル文字をインラインで入力できます。

フォントの設定

使用する言語での表示を正しく行うには、以下の手順でその言語に合ったフォントを選択してください。

1. DOORS Analyst メニューバーで、［ツール］メニューから［オプション］を選択します。
2. ［形式］タブを選択します。
3. ［カテゴリ］を選択して、フォントを指定します。
   - **Dialog fixed font**：固定幅のフォントを使用するダイアグラム向けの設定です。
   - **Developer diagram symbol font**：その他のシンボルとダイアグラムに使用するフォントを設定します。
   - **Report Windows**：出力ウィンドウのタブに使用するフォントを設定します。
   - **Output Windows**：出力ウィンドウの［メッセージ］および［スクリプト］タブに使用するフォントを設定します。
   - **Tcl Files**：DOORS Analyst で開いた Tcl ファイルおよびテキスト ファイルに使用するフォントを設定します。
   - **C/C++ Header/Source**：DOORS Analyst で開いた C/C++ ヘッダ ファイルとソース ファイルに使用するフォントを設定します。

注記

以下の手順はダイアグラムに要素を作成する前に行っておく必要があります。

ダイアグラム要素についてのフォント設定があります。

1. DOORS Analyst メニューバーで、［ツール］メニューから［オプション］を選択します。
2. ［フォント設定］タブを選択します。
3. 374 ページの「［フォント設定］タブ」を参照してフォントタイプを設定します。

注記

［ダイアグラム要素のプロパティ］ツールバーから各要素のフォント スタイルとサイズを変更することもできます。

CJK 文字を使用したモデリング

DOORS Analyst は、CJK 文字を使用したモデリングをサポートしています。以下の要素などに CJK 文字を使用できます。

- すべての要素の名前
- コメント
- Charstring 型リテラル

CJK 文字は英文字と同じように入力できます。CJK 文字を使用してモデルを作成するときに特殊な操作を行う必要はありません。

CJK 文字を使用するための前提条件

DOORS フォーマル モジュールのダイアグラムで CJK 文字が正しく表示されるためには、以下の言語設定が適切に行われている必要があります。

- Unicode 対応でないプログラムの言語
- コード ページ変換テーブル

これらの設定を変更するには以下の手順を行います。

- コントロール パネルで、［地域と言語のオプション］->［詳細設定］->［Unicode 対応でないプログラムの言語］に、使用する言語を指定します。
- ［コード ページ変換テーブル］で、使用する言語のコード ページをすべて選択します。

また、その国の言語に対応するフォントをダイアグラムで使用する必要があります。

新規ダイアグラムのデフォルト フォントを変更するには、以下の手順を行います。

- ［ツール］->［オプション］->［形式］で、以下のフォントを変更します。

```
Developer diagram symbol font (for normal symbols)
Developer diagram code font (for fixed text symbols)
```

注記 1

ここで設定されたフォントは、新規に作成するダイアグラムにのみ適用されます。既存のダイアグラムのフォントは、ダイアグラム要素のプロパティ ツールバーから手動で変更する必要があります。

［Developer diagram symbol font］のプロパティは、各ダイアグラムに設定されます。テキスト シンボル、タスク シンボル、コメント シンボルなどの、固定テキストシンボルに設定されます。

注記 2

すべての設定が上記で説明したとおりに行われていない場合、DOORS Analyst ではダイアグラムは正しく表示されますが、DOORS のフォーマル モジュール内では正しく表示されません。

これは、使用フォントがシステムに存在しない場合でも、DOORS Analyst では代替フォントによって正しく表示されるからです。

テキスト ファイルの処理

テキスト ファイルを DOORS Analyst 内で開くことができます。DOORS Analyst は、テキスト ファイルのローカル ANSI エンコーディングと UTF-8 をサポートしています。既存のテキスト ファイルを DOORS Analyst で開いた場合は、DOORS Analyst によってファイルが元のエンコードの状態で保存されます。テキスト ファイルを DOORS Analyst で作成した場合は、デフォルトによりファイルが UTF-8 で保存されます。［名前をつけて保存］ダイアログからエンコード タイプを選択できます。

制限事項

- Shift-JIS の 0x80 ～ 0xFF で定義されている 1 バイトのカタカナと日本語文字はサポートされません。

- CJK 文字はプロジェクト名に使用できません。

10

便利なショートカット キー

このセクションでは便利なショートカット キーについて説明します。他の標準的なアプリケーションと同じようにアクセス キーを使用できます。

ワークスペースの操作

キーボード ショートカット	説明
Ctrl + N 次に Ctrl + Tab で［ワークスペース］タブに移動	新規ワークスペースを作成します。
Ctrl + O	既存のワークスペースを開きます。
テンキーのマイナス（-）記号	選択したエンティティのツリーを畳みます。
テンキーの乗算（*）記号	モデル ツリーを選択したレベルより 1 つ下に展開します。このキーを使用するたびに、さらに 1 つ下のレベルにツリーを展開できます。
テンキーのプラス（+）記号	選択を展開します。
Alt +4	モデル ビューの再構成。モデル フィルターを選択します。

プロジェクトの操作

キーボード ショートカット	説明
Ctrl + N 次に Ctrl + Tab で［プロジェクト］タブに移動	新規プロジェクトを作成します。
Ctrl + O	プロジェクトを開きます。

ファイルの操作

キーボード ショートカット	説明
Ctrl + N	新規ファイルを作成します。
Ctrl + O	ファイルを開きます。
Ctrl + P	アクティブなドキュメントを印刷します。
Ctrl + S	アクティブなドキュメントを保存します。

ファイル内での移動

キーボード ショートカット	説明
Ctrl + 下矢印	挿入ポイントを移動せずに、数行下にスクロールします。
Ctrl + End	挿入ポイントをファイルの最後に移動します。
Ctrl + Shift + G	［指定行に移動］ダイアログを開きます。
Ctrl+ Home	挿入ポイントをファイルの先頭に移動します。
Ctrl + 左矢印	挿入ポイントを 1 語左に移動します。
Ctrl + M	出力ウィンドウの［ナビゲータ］タブを開きます。
Ctrl + 右矢印	挿入ポイントを 1 語右に移動します。
Ctrl + 上矢印	挿入ポイントを移動せずに、数行上にスクロールします。
End	挿入ポイントを行の最後に移動します。
Home	挿入ポイントを行の先頭に移動します。

テキストの選択

キーボード ショートカット	説明
Ctrl + Shift + End	挿入ポイントの位置からファイルの最後までのテキストを選択します。
Ctrl + Shift + Home	挿入ポイントの位置からファイルの先頭までのテキストを選択します。
Ctrl + Shift + 左矢印	挿入ポイントの左の 1 語を選択します。
Ctrl + Shift + 右矢印	挿入ポイントの右の 1 語を選択します。
Shift + 下矢印	挿入ポイントの位置から 1 行下までのテキストを選択します。
Shift + End	挿入ポイントの位置から行の最後までのテキストを選択します。
Shift + Home	挿入ポイントの位置から行の先頭までのテキストを選択します。
Shift + 左矢印	挿入ポイントの左の 1 文字を選択します。
Shift + 右矢印	挿入ポイントの右の 1 文字を選択します。
Shift + 上矢印	挿入ポイントの位置から 1 行上までのテキストを選択します。

テキストの編集

キーボード ショートカット	説明
Ctrl + A	すべてを選択します。
Ctrl + C	コピーします。
Ctrl + F	アクティブ ファイル内で検索します。
Ctrl + H	置換します。
Ctrl + スペースバー Shift + スペースバー	名前の完成。カーソル位置までに現在の名前に合致する定義が検出された場合。複数が合致する場合は、[名前の完成] スクロール メニューが表示されます。
Ctrl + V	貼り付けます。
Ctrl + X	切り取ります。
Ctrl + Y	やり直します。
Ctrl + Z	取り消し

キーボード ショートカット	説明
F1	現在の選択に関するテキスト構文のヘルプを表示します。
Shift + F8	コメントやユーザーが追加したフォーマットを破棄して、モデルからテキストを復元します。
Shift + 矢印キー	現在のテキスト選択範囲を拡大します。このためには現在テキストを選択している必要があります。
Shift + End	挿入ポイントの位置からテキスト行の最後までのテキストを選択します。
Shift + Home	テキスト行の先頭から挿入ポイントの位置までのテキストを選択します。

エディタのショートカット

キーボード ショートカット	説明
矢印キー	矢印の方向にあるシンボルを選択します。現在シンボルを選択している必要があります。
Ctrl + < ダイアグラムへのシンボル配置時にクリック >	同じタイプのシンボルを複数配置できます。シンボル ツールバーから最初にシンボルを 1 つ選択している必要があります。
Ctrl + < 状態機械 フローへのシンボル配置時にクリック >	フローにシンボルを挿入できます。フロー内の 1 つ前のシンボルまたはフローラインを選択している必要があります。
Ctrl + < ダイアグラム内の単語をクリック >	定義に移動します。定義が含まれるダイアグラムがない場合、モデル ナビゲータが開きます。
Ctrl + < 状態機械 フロー内で選択したシンボルをダブルクリック >	フロー内の選択したシンボルから下をすべて選択します。分岐されたフロー（複数シグナル、分岐など）も選択されます。
Ctrl + < ホイールボタンを回転 >	ダイアグラムを水平方向にスクロールします（インテリマウス ポインティング デバイスが必要です）。
Ctrl + Alt + End	ダイアグラム ナビゲーション。ダイアグラム スコープ内で下に移動します。
Ctrl + Alt + Page Down Ctrl + Alt + Tab	ダイアグラム ナビゲーション。ダイアグラム スコープ内で次のダイアグラムに移動します。
Ctrl + 矢印キー	選択したシンボルを矢印の方向に 5 グリッド（目盛り単位）移動します。

キーボード ショートカット	説明
Ctrl + Delete	モデルからの削除。プレゼンテーション要素と対応するモデル要素を削除します。他のプレゼンテーション要素がこのモデル要素に接続されている場合、これらのプレゼンテーション要素も削除されます。
Ctrl + テンキーの除算（/）記号	すべての操作を非表示にします。（クラス、タイマー、シグナル、インターフェイス、操作、状態機械、データ型、列挙などのシグニチャ シンボルに有効です。）
Ctrl + F3	同じモデル要素の次のプレゼンテーション要素に移動します。
Ctrl + テンキーのマイナス（-）記号	すべての属性とパラメータを非表示にします。（クラス、タイマー、シグナル、インターフェイス、操作、状態機械、データ型、列挙などのシグニチャ シンボルに有効です。）
Ctrl + テンキーの乗算（*）記号	すべての操作を表示します。（クラス、タイマー、シグナル、インターフェイス、操作、状態機械、データ型、列挙などのシグニチャ シンボルに有効です。）
Ctrl + テンキーのプラス（+）記号	すべての属性とパラメータを表示します。（クラス、タイマー、シグナル、インターフェイス、操作、状態機械、データ型、列挙などのシグニチャ シンボルに有効です。）
Ctrl + Shift + < ツールバーのシンボルをクリック >	相互作用概観図とアクティビティ図：シンボルの追加と方向の切り替えを行います。シンボルの位置は、現在選択されていない方向になります。（追加するには、シンボルが選択されている必要があります。）
Ctrl + Shift + 矢印キー	選択したシンボルを矢印の方向に 1 グリッド（目盛り単位）移動します。
CTRL + SHIFT + M	［プレゼンテーションの作成］ダイアログを開きます。
Ctrl + Tab	開いている次のダイアグラムに切り替えます。
Ctrl+Alt + Home	ダイアグラム ナビゲーション。ダイアグラム スコープ内で上に移動します。
Ctrl + Alt + Page Up Ctrl + Alt + Shift + Tab	ダイアグラム ナビゲーション。ダイアグラム スコープ内で前のダイアグラムに移動します。
Esc、Delete < キャンバスを右クリック >	ラインの作成を中止します。
F2	選択したシンボルで編集モードに入ります。
F4	出力ウィンドウ内で次の選択に移動します。
Shift + < ツールバーのシンボルをクリック >	ダイアグラムでシンボルを作成して追加します。自動作成されたシンボルはグレー表示されます。（追加するには、シンボルが選択されている必要があります。）

キーボード ショートカット	説明
Shift + 矢印キー	矢印の方向にあるシンボルを選択して、選択範囲に含めます。（現在シンボルを選択している必要があります。）
Shift + F4	出力ウィンドウウィンドウ内で前の選択に移動します。
Alt + 上矢印キー	モデル ビューで選択したノードを上に移動します。
Alt + 下矢印キー	モデル ビューで選択したノードを下に移動します。
Shift + Enter	モデル ビューで選択したダイアグラム要素のモデル要素を表示します。
F8	現在の選択を確認します。.
Ctrl + F8	現在の選択を確認しません。
Shift + スペースバー	自動作成。現在の選択に対して自動作成できる要素がすべて表示されます。自動配置を参照してください。
Ctrl + スペースバー	自動挿入。現在の選択の後ろに自動挿入できる要素がすべて表示されます。自動配置を参照してください。

ウィンドウ ナビゲーション

キーボード ショートカット	説明
Alt +1	全画面表示に切り替えます。
Ctrl + F2	カーソル位置の定義をモデル ナビゲータの［お気に入り］に入れます。
Ctrl + F4	アクティブ ウィンドウを閉じます。
Ctrl + Shift + Tab Ctrl + Shift + F6	前のウィンドウに移動します。
Ctrl + Tab Ctrl + F6	次のウィンドウに移動します。
Shift + F2	カーソル位置の定義のコンテキストにしたがって、モデル ナビゲータを表示します。

プロパティ エディタ

キーボード ショートカット	説明
Alt + Enter	プロパティ エディタを表示します。
Ctrl + BackSpace	所有者に移動。モデル ツリーのスコープを現在の選択の所有者に変更します。
Ctrl + Alt + C	コントロール ビューに切り替えます。
Ctrl + Alt + T	テキスト ビューに切り替えます。

ウィンドウとダイアログの表示／非表示

キーボード ショートカット	説明
Alt +0	ワークスペース ウィンドウを表示／非表示にします。
Alt +2	出力ウィンドウを表示／非表示にします。
Alt + Enter	プロパティ エディタを表示します。
CTRL + Q	選択に対して［クエリ］ダイアログを開きます。
F1	ヘルプを表示します。

ズーム / パン

キーボード ショートカット	説明
< ホイール ボタンを回転 >	ダイアグラムを垂直方向にスクロールします（インテリマウス ポインティング デバイスが必要です）。
< ホイール ボタンをダブルクリック >	100%表示にします。
Shift +< ホイール ボタンをダブルクリック >	エディタ ウィンドウに全体を表示します。
Shift + < ホイール ボタンを回転 >	ホイールの回転方向に従ってズーム イン／アウトします。ズーム インの中心はマウス ポインタの位置です。
CTRL + Shift + < ホイール ボタンを回転 >	1 本のラインを選択している場合、ダイアグラムがそのラインに沿ってスクロールします。どちらかのエンドポイントが表示の中央になるとスクロールが停止します（インテリマウス ポインティング デバイスが必要です）。

キーボード ショートカット	説明
テンキーのマイナス（-）記号	25% ズームアウトします。（ダイアグラムがアクティブの場合、またどの要素でもテキスト編集モードになっていない場合に有効です。）
テンキーのプラス（+）記号	25% ズームインします。（ダイアグラムがアクティブの場合、またどの要素でもテキスト編集モードになっていない場合に有効です。）
左不等号（<）	1 本のラインを選択している場合、そのラインのソース エンドポイントの方向にダイアグラムがスクロールします。
右不等号（>）	1 本のラインを選択している場合、そのラインのデスティネーション エンドポイントの方向にダイアグラムがスクロールします。

11

ダイアログ ヘルプ

このセクションでは、ダイアログのヘルプ ボタンをクリックすると表示されるヘルプ テキストについて説明します。

［新規］ウィザード

［ファイル］タブ

このダイアログで、新しいファイルを追加できます。

- ファイルの追加時に、ファイル名と場所を指定する必要があります。
- ファイルは既存のプロジェクトに追加できます。ファイルを追加するには、このプロジェクトを［ファイル ビュー］で開く必要があります。
- 新しいファイルがデスクトップに開きます。

［プロジェクト］タブ

このダイアログで、新しいプロジェクトを追加できます。

プロジェクトを追加する際、プロジェクトの使用方法を指定します。以下に示すように、選択に従って起動時に異なるアドインがロードされます。

UML（モデリング用）
アドインはロードされません。

- プロジェクトの追加時に、プロジェクト名と場所を指定する必要があります。
- プロジェクトは現在のワークスペースに入れるか、あるいはプロジェクトのために新しいワークスペースを作成します。

UML プロジェクト - 2 ページ目

このダイアログは、モデルを含むファイル用に推奨されるファイル ディレクトリと名前を表示します。

- この推奨値は変更するか、そのまま確定できます。
- オプションとして、空のパッケージを追加できます。

UML プロジェクト - 3 ページ目

このダイアログは、プロジェクト名と関連ファイルの名前を表示します

- ［完了］ボタンをクリックして名前を確定するか、［戻る］ボタンをクリックして変更できます。
- 新しいプロジェクトがワークスペース ウィンドウに表示されます。

ワークスペース

このダイアログで、新しいワークスペースを追加できます。

- ワークスペースの追加時に、ワークスペース名と場所を指定する必要があります。
- 新しいワークスペースは、ワークスペース ウィンドウにロードされます。

［カスタマイズ］ダイアログ

［コマンド］タブ

このタブには、ツールバーのボタンとともにデフォルト メニューやコマンド、およびメニューが一覧表示されます。これらのコントロールはツールバーやメニューに追加できます。このタブで、ツール バーのボタンを移動、追加、削除できます。

1. ［カテゴリ］ボックスで、カスタマイズするツールバーの名前をクリックします。
2. ［ボタン］領域で、ダイアログからツールバーに項目をドラッグします。最初に項目をクリックして、特定の項目に関する情報を表示します。
3. ツール バーから項目を削除するには、ツールバーからダイアログに項目をドラッグします。

ツール バーにボタンを追加するには

1. 変更しようとするツール バーが表示されていることを確認します。
2. ［カテゴリ］ボックスに、使用可能なツール バー ボタンや項目がグループ化されています。追加するツール バー ボタンや項目があるカテゴリを選択します。
3. ボタンや項目をクリックして、機能に関する情報を表示します。
4. ［ボタン］領域からユーザー インターフェイスのツール バーにボタンや項目をドラッグします。

ツール バーからボタンを削除するには

1. 削除しようとするツール バーが表示されていることを確認します。
2. ツール バーからボタンや項目をドラッグして削除します。

デフォルトのボタンをツール バーから削除しても、［カスタマイズ］ダイアログにはそのボタンが残ります。表示をカスタマイズしたツール バーボタンを削除すると、その表示は完全になくなります。ただし、コマンドは［カスタマイズ］ダイアログの［コマンド］タブから使用可能です。

ヒント

表示をカスタマイズしたツール バー ボタンを再利用のため保存するには、未使用のボタンを格納するツール バーを作成してこのボタンを移動し、格納したツール バーを非表示にします。

［ツール バー］タブ

このタブは、標準のツールバーとカスタムのツール バーを一覧表示します。

チェック ボックスを選択するか、選択を解除して、ツール バーを表示または非表示にします。各ツール バーはデフォルトの場所、または最後にツール バーが移動された場所に表示されます。メニュー バーは非表示にできません。

ツールチップを表示
このチェック ボックスをクリックし、ツール バーのボタンまたはフィールドにカーソルを移動したときツールチップが表示されるようにします。

大きいボタン
このチェック ボックスをクリックし、ツール バーのボタンのサイズを大きくします。

新規ツール バーを作成するには

1. ［新規］をクリックします。
2. 表示されたダイアログに、ツール バーの名前を入力します。新しいツール バーがインターフェイスのツール バー領域に表示されます。
3. ［コマンド］タブで、ツール バーに追加する項目を選択します。

デフォルトのツール バー設定を復元するには

1. リストのツール バーをクリックします。
2. ［リセット］をクリックします。

ユーザーが作成したツール バーは復元できません。

ユーザーが作成したツール バーを削除するには

1. リストのツール バーをクリックします。
2. ［削除］をクリックします。

デフォルトのツール バーは削除できません。

ユーザーが作成したツール バーの名前を変更するには

1. リストのツール バーをクリックします。
2. ［ツールバー名］フィールドにツール バーの新しい名前を入力します。
3. もう一度ツール バーをクリックして、変更を保存します。

新規ツールバーの作成

新しいカスタム ツール バーの名前を入力します。大文字または小文字を使用できますが、大文字／小文字に関係なく名前は一意でなければなりません。他のツール バーと同じ名前にはできません。この名前を後で変更する場合は、［ツール バー］タブの［ツールバー名］フィールドで名前を編集できます。

ウィンドウのレイアウト

このタブでウィンドウのレイアウトをカスタマイズできます。ドッキング ウィンドウのツール バーの位置、表示、場所を保存できます。

新規レイアウトの保存

1. ［新規］ボタンをクリックします。
2. レイアウトに付ける名前を入力します。

3. ウィンドウを閉じます。

新規レイアウトを復元するには

1. 復元するレイアウトをクリックします。
2. ［復元］をクリックします。

レイアウトを削除するには

1. 削除するレイアウトをクリックします。
2. ［削除］ボタンをクリックします。

［ツール］タブ

このタブで、［ツール］メニューにコマンドを追加できます。これらのコマンドをオペレーティング システムで実行されるプログラムに関連付けることができます。この情報は、以下のディレクトリの Tools.dat というファイルに保存されます。

```
C:¥Documents and Settings¥<user>¥Application
Data¥Telelogic¥Shared
```

［ツール］メニューにコマンドを追加するには

1. ［新規（挿入）］ボタンをクリックします。空の矩形で示される空白行が、［メニューの内容］ボックスに表示されます。
2. ［ツール］メニューに表示されるように、コマンドの名前を入力します。**Enter** キーを押して、名前を保存します。
3. ［コマンド］フィールドにプログラムへのパスを入力します。参照ボタンをクリックしてプログラムの場所を検索することもできます。
4. ［引数］テキストボックスに、プログラムに渡す引数を参照または手動で入力します。［引数］テキストボックスの隣のドロップダウン ボタンをクリックして、使用できる引数のリストを表示できます。
5. ［初期ディレクトリ］ボックスで、コマンドを挿入するファイル ディレクトリを参照または入力します。
6. プログラムがコンソール プログラム（Windows コマンド プロンプトなど）であれば、出力ウィンドウで実行するようにできます。このためには、［出力ウィンドウを使う］チェック ボックスを選択します。
7. コマンドを使用するたびに引数を変更できるよう設定する場合、［引数を要求する］チェック ボックスを選択します。
8. アプリケーションの出力を OEM 形式にする場合、［OEM 形式を使う］チェック ボックスを選択します。
9. ［OK］をクリックします。［ツール］メニューにコマンドが表示されます。

その他の作業

- サブメニューにコマンドを挿入するには、メニュー名とコマンド名を円記号「¥」で区切ります。たとえば、エディタ メニューのコマンド「メモ帳」は「editor¥Notepad」と入力します。
- アクセス キーを挿入するには、名前内で選択する文字の前にアンパサンド「&」を入力します。
- ［上に移動］ボタンと［下に移動］ボタンを使用して、メニュー内でコマンドを上下に移動します。
- コマンドの名前を変更するには、コマンドをダブルクリックして、新しい名前を入力します。

［ツール］メニューからコマンドを削除するには

1. リストのコマンドをクリックします。
2. ［削除］ボタンをクリックします。

注記

DOORS Analyst のスコープでアドインを作成することは推奨できません。

参照

第 55 章「Telelogic Tau のカスタマイズ」の 1697 ページ、「アドインの内容と構造」

［オプション］ダイアログ

［一般］タブ

このタブで一般的なオプションを設定できます。

［ステータスバーを表示する］
DOORS Analyst ユーザー インターフェイスの下部にあるステータス バーの表示、非表示を切り替えられます。

［内容の受信時に出力ウィンドウを表示する］
出力ウィンドウを閉じると、通常このウィンドウのさまざまなタブに現れる情報が表示されなくなります。しかし、このオプションを選択すると、手動でチェックを行った後など、新しい情報を表示する必要がある場合に、出力ウィンドウが自動的に開きます。

印刷マネージャで選択部分をトラックする
印刷マネージャはデフォルトでモデルビューのアクティブな選択部分をトラックします。このオプションを選択すると、このトラッキングが無効になります。

詳細オプションページを表示する
このチェック ボックスを選択すると、すべてのオプションをツリー構造で表示する［詳細］タブが表示されます。いくつかのオプションはこの詳細オプション表示でのみ表示されます。

タブ付きドキュメント
このチェックボックスを選択すると、1 つのウィンドウにドキュメントがタブとして表示されます。

ウェルカムページを起動時に表示する
このオプションはツール起動時のウェルカムページの表示 / 非表示を制御します。このオプションはウェルカムページからも設定できます。オフにした場合は、ウェルカムページは［ヘルプ］メニューから開いてください。

ソースコントロールプロバイダ
ソース コントロール システムをインストールしている場合、このチェックボックスを選択できます。このチェック ボックスを選択すると、ソース コントロール メニューとツール バーを通じた、ソース コントロール システムとの相互作用が可能になります。詳細は 構成管理 を参照してください。

ファイルの自動更新
このオプションは、ジェネリック ソースコントロールをソースコントロールプロバイダとして選択している場合に使用できます。このオプションが有効になっている場合は、ファイルは、チェックアウト前に自動的に CM システムから更新されます。

次の種類のファイルに対する外部プログラムの起動を無効にする
このフィールドでファイルの拡張子を指定し、これらのファイルを DOORS Analyst から開いたとき、関連付けられている外部アプリケーションが起動しないように設定できます。たとえば、`.txt` 拡張子を追加すると、DOORS Analyst からテキストファイルを開いた場合、外部のテキストエディタではなく DOORS Analyst のテキストエディタでファイルが開かれます。

デフォルトのヘルプ コンテキストを選択する
Telelogic の複数のツールをインストールしている場合、［デフォルトのヘルプ コンテキストを選択する］ボックスで、デフォルトとして使用するヘルプ ファイルを選択できます。

URN Map
URN（Universal Resource Name）Map を使用して、ファイル保存場所の短縮名を定義できます。

例：

```
home:C:¥MyHomeDir;work:C:¥MyWorkDir
```

「home」は `C:¥MyHomeDir` の短縮名、「work」は `C:¥MyWorkDir` の短縮名です。ユーザーは独自の環境の URN を定義できます。定義した短縮名は、コンポーネントがファイル、ビットマップなどのリソースを参照する際に使用されます。

保存

このタブで、DOORS Analyst の保存オプションを設定できます。

ツールを実行する前に保存する
外部ツールが起動する前に未保存の作業をすべて自動保存するよう設定します。

ファイルとプロジェクトを保存する前に尋ねる
エディタを閉じる際、ファイルやプロジェクトの保存プロンプトを表示するよう設定します。

外部で修正されたファイルを自動的に再ロードする
デフォルトで、情報メッセージが表示され、外部修正したファイルを再ロードするよう促されます。このオプションを選択すると、DOORS Analyst 以外のツールで修正したファイルが自動的に再ロードされます。

すべてのロードされたプロジェクトのアドイン状態を保存する
ロードされているアドインを、現在ロードしているすべてのプロジェクトでアクティブにします。

自動バックアップ
［有効にする］チェック ボックスを選択し、あらかじめ設定した間隔でモデルを自動保存するよう設定します。数字を入力するか上下ボタンをクリックして、任意の保存間隔（分単位）を入力できます。

［ワークスペース］タブ

このタブで、開いているワークスペースに一般的なオプションを設定できます。

起動時に前回のワークスペースを再ロードする
DOORS Analyst を最後に使用した際に作業していたワークスペースを開くように設定します。

プロジェクト ファイルのステータス変更時に警告を出す
作業中のプロジェクト ファイルの状態が読み取り専用に変更された場合、警告を受け取るように設定します。これで未保存の作業を消失する危険を回避できます。

プロジェクトのデフォルトの場所
新規プロジェクトの作成時に、プロジェクト ファイルの保存場所が表示されるように設定します。このテキスト フィールドで、新規プロジェクトを保存するフォルダのパスを入力または参照できます。

［形式］タブ

このタブで、ウィンドウとファイル内のテキストと色の表示を調整できます。

カテゴリを選択すると、以下を選択できます。

- ファイルやウィンドウ内のテキストの**フォント**と**サイズ**
- 選択したカテゴリの背景色とテキストの色。デフォルトで、コントロール パネルで定義している配色が使用されます。［自動］チェック ボックスの選択を解除すると、テキストと背景色を設定できます。

［フォント設定］タブ

このタブで、新規ダイアグラムを作成する際に使用するデフォルトのフォントを選択できます。

ダイアグラムフォント設定
このフォント設定では、通常のダイアグラム要素のデフォルトの表示を決定します。

固定フォント設定
このフォント設定では、固定幅フォントで表示したほうが見映えのいいテキストを持つシンボルのデフォルトの表示を決定します。このフォント設定が適切であるシンボルの例として、テキストシンボルがあります。［有効］チェックボックスをチェックすると、この種類のシンボルを作成したときに、固定幅のフォントが適用されます。

ラベルフォント設定
このフォント設定は、ダイアグラム要素の主たるラベル以外のテキストラベルのデフォルトの表示を決定します。例としては、クラスシンボルの属性や操作のラベルがあげられます。［有効］チェックボックスをチェックすると、この種類のラベル要素にこのフォント設定が適用されます。

［リンク］タブ

このタブで、リンク作成の振る舞いをカスタマイズできます。

修正されたオブジェクトからアクティブなリンク端へのリンクにする
このオプションを選択せずに、リンクの自動作成を使用すると、アクティブなリンク端から他のモデルへのリンクを作成できます。このオプションを選択すると、他のモデルからアクティブなリンク端へのリンクが生成されます。

修正されたオブジェクトとアクティブなリンク端とのリンクを自動作成
このオプションを選択して、アクティブな 1 つのリンク端を選択すると、すべての修正内容がこのリンク端にリンクされます。

リンク インジケータを表示
このオプションを選択すると、DOORS Analyst でリンク マーカーが表示されます。

ドラッグ & ドロップでリンクを作成するときに、要求をターゲットとして使用
リンクはドラッグ & ドロップ操作で作成できます。このオプションが選択されていると、リンクのターゲットは要求になります。このオプションが選択されていない場合は、要求はリンクのソースになります。

Web server

Studio - Settings - WebServer

PortRangeBegin オプションと **PortRangeEnd** オプションは、Tau Web サーバーが使用する TCP/IP ポートの範囲を定義します。Tau を使うマシンでこれらのデフォルトのポート番号がすでに使用されている場合は、ここで値を変更します。

Proxy settings

U2 - Options - ProxySettings

Host、**Password**、**User** の各オプションは HTTP プロキシーサーバーを通して Web にアクセスする場合に設定します。設定すると Tau から URL を使って情報にアクセスする際に常にプロキシーサーバーが使用されます。たとえば、WSDL ファイルをある URL からインポートするような場合などです。Host オプションの形式は、`<address>:<port>` です。

エディタのショートカット

要素の表示

このダイアログで、既存のモデルから一括で選択したシンボルを、別のダイアグラムに追加できます。

- 要素リストのチェック ボックスをチェックして、複数の要素を選択できます。
- この要素リストには現在設定されているスコープの要素が含まれます。
- ［スコープの選択］ボタンを使用して、モデル内の任意のスコープから要素をリストに追加できます。

あらかじめ要素を選択してこのダイアログを表示した場合、リスト内でその要素がすでに選択されています。

- 新しいダイアグラムがデスクトップに開きます。

モデル ビューの再構成

使用するブラウザ モデルを選択します。あらかじめ定義されたブラウザ ビューが 2 つあります。［Standard View］は、設計の詳細を含んだロードされたモデルの総合的なビューを提供します。このビューは、デザイン指向のユーザーに適しています。

もう 1 つの［Diagram View］は、ロードされたモデルの簡単なビューを提供します。このビューは、設計指向のユーザーに適しています。

参照

第 i 章「UML 言語ガイド」の 273 ページ、「メタモデル」

その他のオプション

ステレオタイプ

要素に適用するステレオタイプを選択します。それぞれの行をクリックすると、各ステレオタイプの説明が表示されます。適用できるステレオタイプの数は選択した要素によって異なります。

参照

第 i 章「UML 言語ガイド」の 274 ページ、「ステレオタイプ」

12

その他のリソース

このセクションでは、ヘルプ ファイル以外で DOORS Analyst に関する知識を広げるために役立つドキュメントについて説明します。役に立つ Web へのリンクも提供しています。

リンク

サポートへのお問い合わせ

Telelogic 製品のサポートと情報は、Telelogic サポートサイトから IBM Rational Software Support に移行中です。この移行期間中は、サポートの連絡先がお客様によって異なります。

製品サポート

- 2008 年 11 月 1 日より前に Telelogic 製品を取引されたお客様は、サポート ウェブサイト DOORS Analyst Web site をアクセスしてください。製品情報の移行後に、IBM Rational Software Support site に自動で転送されます。
- 2008 年 11 月 1 日より前に Telelogic 製品のライセンスをお持ちではなかった新規のお客様は、IBM Rational Software Support site をアクセスしてください。

お客様サポートにお問い合わせいただく前に、問題を説明するために必要な情報をご用意ください。IBM ソフトウェアサポート担当員に問題を説明する際には、担当員が迅速に問題を解決できるように、問題の具体的な内容と必要な背景情報をすべて伝えてください。あらかじめ以下の情報をご用意ください。

- 問題発生時に使用していたソフトウェアとそのバージョン
- 問題に関連したログ、トレース、メッセージなど
- 問題を再現できるかどうか。再現できる場合はその手順
- 回避策があるかどうか。ある場合は、その回避策の内容

その他の情報

Rational ソフトウェア製品、ニュース、イベント、その他の情報については、IBM Rational Software Web site をご覧ください。

UML ドキュメント

- **UML チュートリアル**

  このチュートリアルの目的は、DOORS Analyst の機能と UML 言語について理解することです。チュートリアルは、DOORS の要件モジュールの基本操作を理解し、UML の基礎知識があるユーザーを対象としています。

  チュートリアルは、インストール ディレクトリの locale\etc に tutorial.pdf の名前であります。

- **UML クイック リファレンス ガイド**

  このドキュメントでは UML で一般的に使用されているグラフィックとテキスト構成要素の例を示します。

  クリック リファレンス ガイドはインストール ディレクトリの locale\etc に quickref.pdf の名前であります。

その他のリンク

Cygwin

Cygwin の各バージョンの正確な内容については以下の Web サイトをご覧ください。
http://www.cygwin.com

GNU C/C++

GNU Compiler Collection でサポートされている C/C++ です。
http://www.gnu.org/software/gcc

ITU-T

旧 CCITT
http://www.itu.int/

Macrovision

FLEXnet または Macrovision の詳細については以下の Web サイトをご覧ください。
http://www.macrovision.com

MISRA

AgileC コード ジェネレータで生成されたコードは、2004 年 10 月より、「MISRA-C:2004 Guidelines for the use of the C language in critical systems」に記述されている MISRA コーディング ルールにほぼ適合しています。以下の Web サイトをご覧ください。

http://www.misra.org.uk

OCL

OCL (Object Constraint Language）の詳細については以下の Web サイトをご覧ください。
http://www.omg.org

OMG

Object Management Group（OMG）の詳細については以下の Web サイトをご覧ください。
http://www.omg.org

PDF

PDF ファイルを読むには Adobe Acrobat Reader を使用します。
www.adobe.com

Tcl

詳細については以下の Tcl Developer の Web サイトをご覧ください。
http://tcl.activestate.com/

TTCN-3

TTCN-3 標準は以下の Web サイトからダウンロードできます。
http://www.etsi.org

XML

Extensible Markup Language（XML）の詳細については以下の Web サイトをご覧ください。
http://www.w3.org/XML

索引

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P

S

T

U

W

X

あ

い

う

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

は

ひ

ふ

へ

ほ

め

も

や

ゆ

よ

ら

り

れ

わ