2017 情財第 280 号

STAMP 支援ツールの開発

マニュアル

開発者向けマニュアル

平成３０年３月

株式会社チェンジビジョン

目　　次

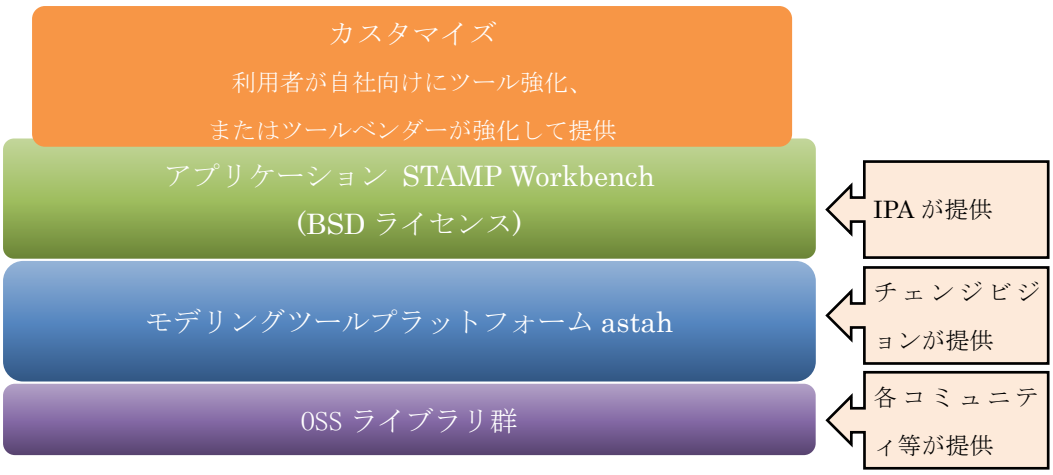
改訂履歴

変更日	Version	変更内容・理由
2018/03/01	1.0	新規作成

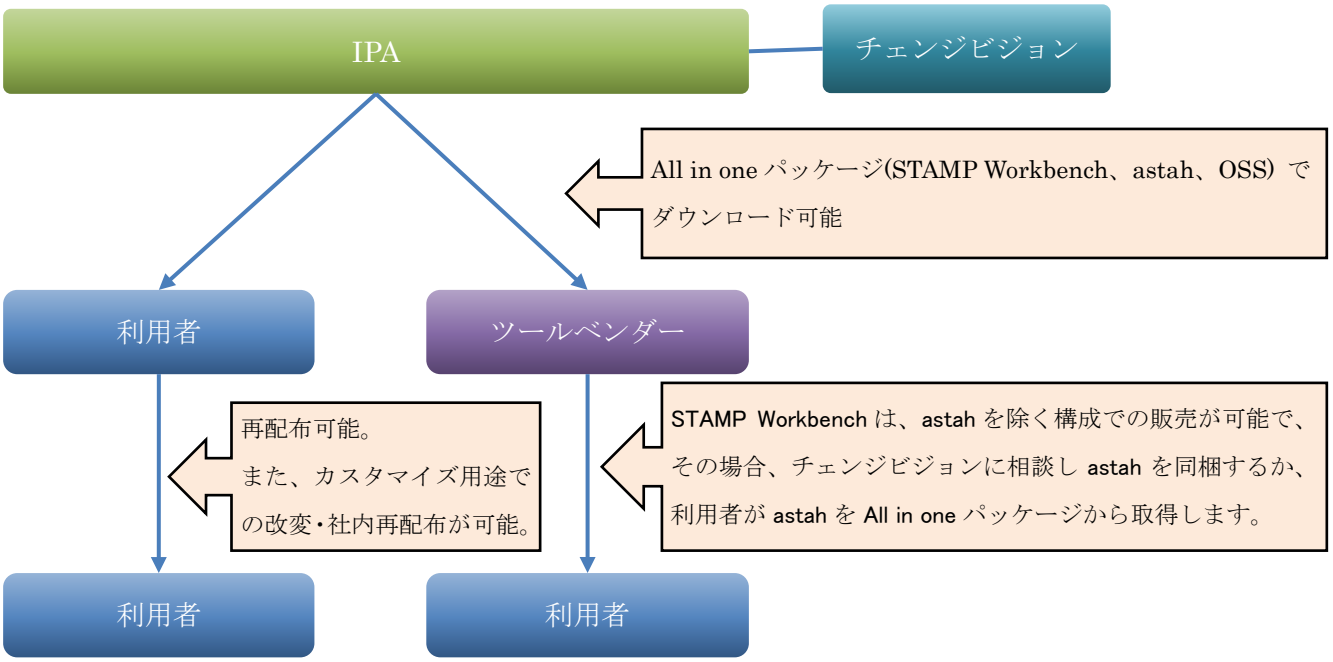
1　STAMP Workbench のカスタマイズと利用

STAMP Workbench のカスタマイズに関連して、できることの概要をここで説明します。権利や禁止事項等の詳細については、各構成要素のライセンスファイルを確認してください。

開発者は、STAMP Workbench をカスタマイズすることが可能です。その場合、アプリケーション STAMP Workbench のソースコードを改変・追加拡張して、使用・再配布が可能です。その用途に限ってモデリングツールプラットフォーム astah の使用・再配布も可能です。



構成に関する概念図

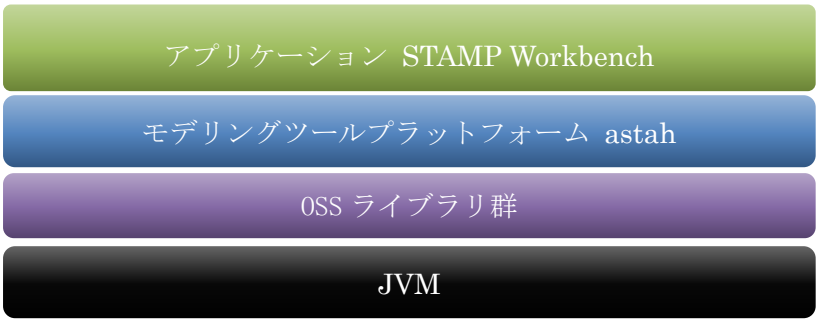


利用に関する概念図

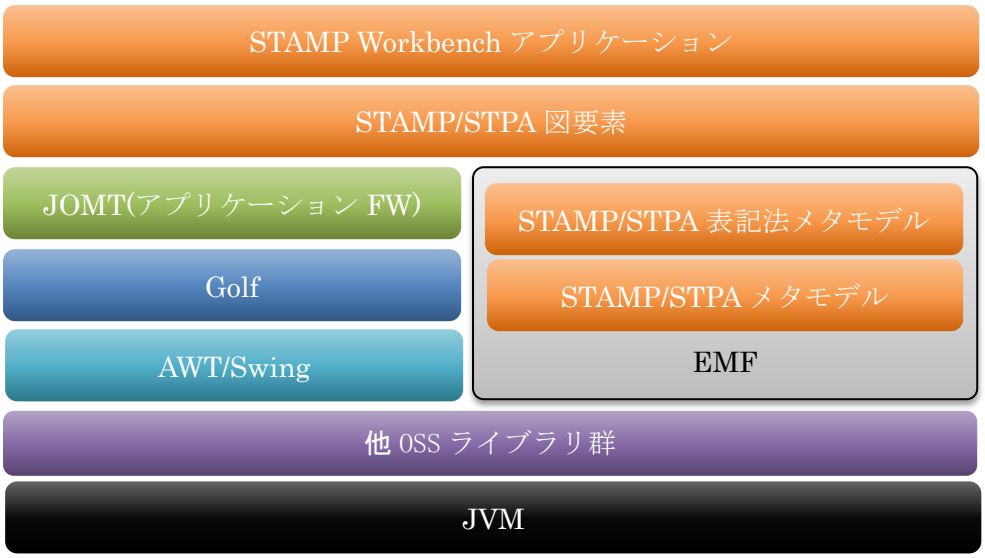
2　アーキテクチャ概要

2.1　　階層化アーキテクチャ

STAMP Workbench のプラットフォームは次のような階層化アーキテクチャを採用しています。



Java 仮想マシン(JVM)上で動作するモデリングツールのプラットフォームが実装されており、そのプラットフォーム上で動作するアプリケーションとして STAMP Workbench が実装されています。



上記は先に示した階層構造を一段階詳細化したもので、プラットフォーム自体の構成および STAMP Workbench がプラットフォームへの拡張として提供するコンポーネント群を示したものです。以下ではこの図で登場するプラットフォームのコンポーネントの概要を説明します。

2.1.1　　JVM および EMF

STAMP Workbench のプラットフォームは Java 仮想マシン(JVM)上で動作し、メタモデル実装やトランザクション管理などモデリングツールとしての基盤には Eclipse Modeling Framework (以下、EMF) を利用しています。

2.1.2　　Golf

Golf は、AWT/Swing の上層に位置するフレームワークで、UI の実装構造やグラフィカルエディタとしての図形の描画などをサポートします。

2.1.3 JOMT

プラットフォームのアプリケーションとしての実装です。各種メタモデルや UI 要素を組み込める構造になっており、STAMP Workbench 固有の実装を差し込むことでツールが成立しています。

2.2 GUI アーキテクチャ

STAMP Workbench の GUI アーキテクチャは Model View Controller (MVC)をベースとしたアーキテクチャを採用しており、MVC のバリエーションの中では、Presentation Model に近い構成になっています。

2.2.1 モデル層

モデル層の実装はモデルの整合性を維持した変更をトランザクションで保護(整合性の確認や Undo/Redo)し、永続化をサポートします。

ドメインモデル

アプリケーションが対象とするドメインをモデル化します。STAMP Workbench では STAMP/STPA の分析モデルの要素である Component や UnsafeControlAction (UCA)を表現するモデルです。

プレゼンテーションモデル

ドメインモデルを表記法に従ってエディタ上で表示するための表示状態をモデル化します。STAMP Workbench ではコントロールストラクチャー図や UCA 表などを表現するモデルです。

2.2.2 ビュー層

ビューエレメント

ビューエレメントは、対応するプレゼンテーションモデルを図上に表示するために描画ツリーを構築および、それぞれのモデルの変更を検出した際には描画ツリーへの反映を行います。

描画(GNode)

GNode は四角形やテキスト、直線、曲線など、図を描画するための図形の描画ツリーを構成します。

Observer パターン

GoF デザインパターンのひとつで、オブジェクトが送信するイベントを監視して処理を行うものです。MVC アーキテクチャの方針に従い、ビューとモデルの依存関係はビューからモデルへの一方向のみとなっており、モデルが変更された際のビューの更新は、ビューがモデルの変更を監視することで行っています。

2.2.3　コントローラ層

モードとコマンド

モードは、選択モードや図要素作成モードといった、図上の編集状態を実装するオブジェクトで、スタック構造を持ち、マウスやキーボード操作を受けて、モデル変更を行うコマンドを構築する責務を負います。

Command パターン

GoF デザインパターンのひとつで、動作とそれに伴うパラメータをカプセル化したものです。モードやコンテキストメニューなどでは、モデルなどに対して作用するコマンドのパラメータをユーザー入力から収集し、コマンドを構築して実行します。

3　STAMP Workbench のモジュール構造

ここでは STAMP Workbench を構成するモジュールと配置されているパッケージや主要なクラスを説明します。

3.1　net.astah.stpa.stamp

STAMP/STPA のメタモデルを実装するモジュールです。

パッケージまたはフォルダ	概要
model/	STAMP/STPA メタモデルの UML モデルや Ecore モデルを含むフォルダです。
net.astah.stpa.stamp	STAMP/STPA メタモデルのパッケージやメタクラスなどの公開インタフェースを含みます。
net.astah.stpa.stamp.impl	STAMP/STPA メタモデルの実装クラスを含みます。
net.astah.stpa.stamp.util	STAMP/STPA メタモデルに関連する Switch クラスや Resource 実装を含みます。
net.astah.stpa.stamp.constraints	STAMP/STPA メタモデルの EMF Validation を用いた検証ルールを実装します。

3.2　net.astah.stpa.stamp.edit

STAMP/STPA のメタモデルの編集をサポートするモジュールで、当該メタモデル用の AdapterFactory および ItemProviderAdapter の実装が中心です。

パッケージまたはフォルダ	概要
net.astah.stpa.stamp.provider	STAMP/STPA メタモデル向けに EMF.Edit がコード生成した各種プロバイダ実装を含みます。
net.astah.stpa.stamp.edit.sequencer	STAMP/STPA モデルの ID 自動採番ルールを実装します。

3.3　net.astah.stpa.stamp.notation

STAMP/STPA の表記法メタモデルを実装するモジュールです。

パッケージまたはフォルダ	概要
model/	STAMP/STPA の表記法メタモデルの UML モデルや Ecore モデルを含むフォルダです。
net.astah.stpa.stamp.notation	STAMP/STPA の表記法メタモデルのパッケージやメタクラスなどの公開インタフェースを含みます。
net.astah.stpa.stamp.notation.impl	STAMP/STPA の表記法メタモデルの実装クラスを含みます。
net.astah.stpa.stamp.notation.util	STAMP/STPA の表記法メタモデルに関連する Switch クラスや Resource 実装を含みます。

3.4 net.astah.stpa.stamp.notation.edit

STAMP/STPA の表記法メタモデルの編集をサポートするモジュールで、当該メタモデルの AdapterFactory および ItemProviderAdapter の実装が中心です。

パッケージまたはフォルダ	概要
net.astah.stpa.stamp.notation.provider	STAMP/STPA の表記法メタモデル向けに EMF.Edit がコード生成した各種プロバイダ実装を含みます。
net.astah.stpa.stamp.notation.edit.command	STAMP/STPA の表記法モデルの初期化やモデル同期などの処理を行うコマンドを実装します。
net.astah.stpa.stamp.notation.edit.layout	STAMP/STPA の表記法モデルのレイアウトに関する処理を実装します。
net.astah.stpa.stamp.notation.edit.reactor	図要素に関係するモデルに変更があった際に同期処理のトリガーを実装します。

3.5 net.astah.stpa.stamp.notation.editor

STAMP/STPA のビューエレメント実装するモジュールです。

パッケージまたはフォルダ	概要
net.astah.stpa.stamp.notation.editor	STAMP/STPA の図要素に対応するビューエレメントやエディタ上での図要素に関係する処理を実装します。

3.6 net.astah.stpa.stamp.ui

STAMP Workbench のエディタや表の UI を実装するモジュールです。

パッケージまたはフォルダ	概要
net.astah.stpa.stamp.ui.control	STAMP/STPA 固有の UI から実行するコマンドを実装します。これらのコマンドは一般的に EMF コマンドでのモデル変更処理に委譲します。
net.astah.stpa.stamp.ui.control.mode	STAMP/STPA 固有の図のモードを実装します。
net.astah.stpa.stamp.ui.menu.handler	STAMP/STPA 固有のメニューハンドラを実装します。
net.astah.stpa.stamp.ui.table	STAMP/STPA の各種表(前提条件表など)を実装するための汎用表の拡張を実装します。
net.astah.stpa.stamp.ui.view	STAMP/STPA の実装で共通的に利用する UI 部品や分析手順ビューなどを実装します。
net.astah.stpa.stamp.ui.view.swing	STAMP/STPA の各種表などの Swing を前提とした処理を実装します。
net.astah.stpa.stamp.ui.proptab	STAMP/STPA 固有のプロパティビュー機能を実装します。

3.7 net.astah.stpa.stamp.app

STAMP Workbench の起動可能なアプリケーションを実装するモジュールです。main()を持ったクラスや検証ルールを有効化する定義などがありますが、基本的に実装は持たずに各モジュールを束ねる役割

を持ちます。

4　STAMP/STPA データ仕様

4.1　　概要

STAMP/STPA のデータモデルは、UML のクラスモデルの形で設計されたメタモデル(モデル構造を規定するモデル)として定義されており、STPA の分析モデルを表現する「STAMP/STPA メタモデル」と、STAMP の図や表を表現する「STAMP/STPA 表記法メタモデル」に分かれています。

また、メタモデルは EMF によるコード生成が可能な UML モデルとして定義されており、UML モデルから EMF のメタメタモデルである Ecore モデルに変換することで、アプリケーションが利用するメタモデルの Java クラスを EMF にコード生成させたものをベースに開発しています。

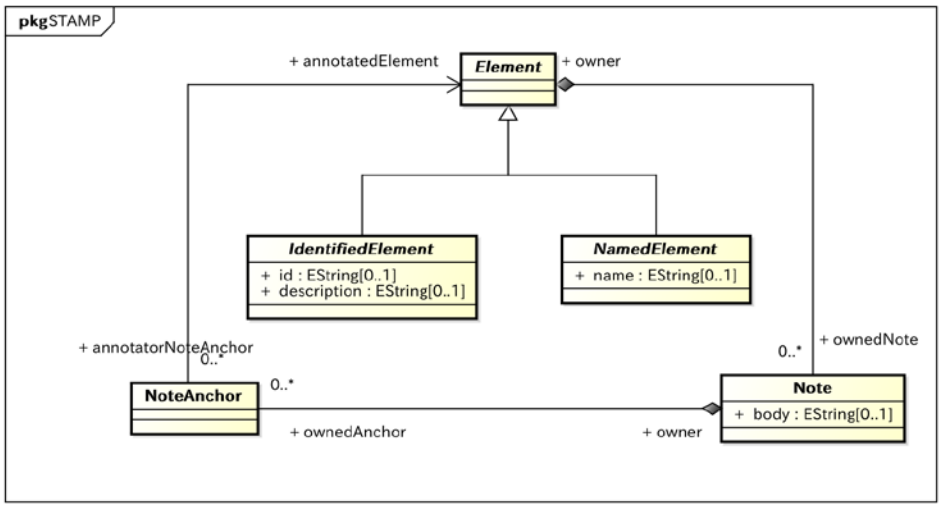
4.1.1　　永続化形式

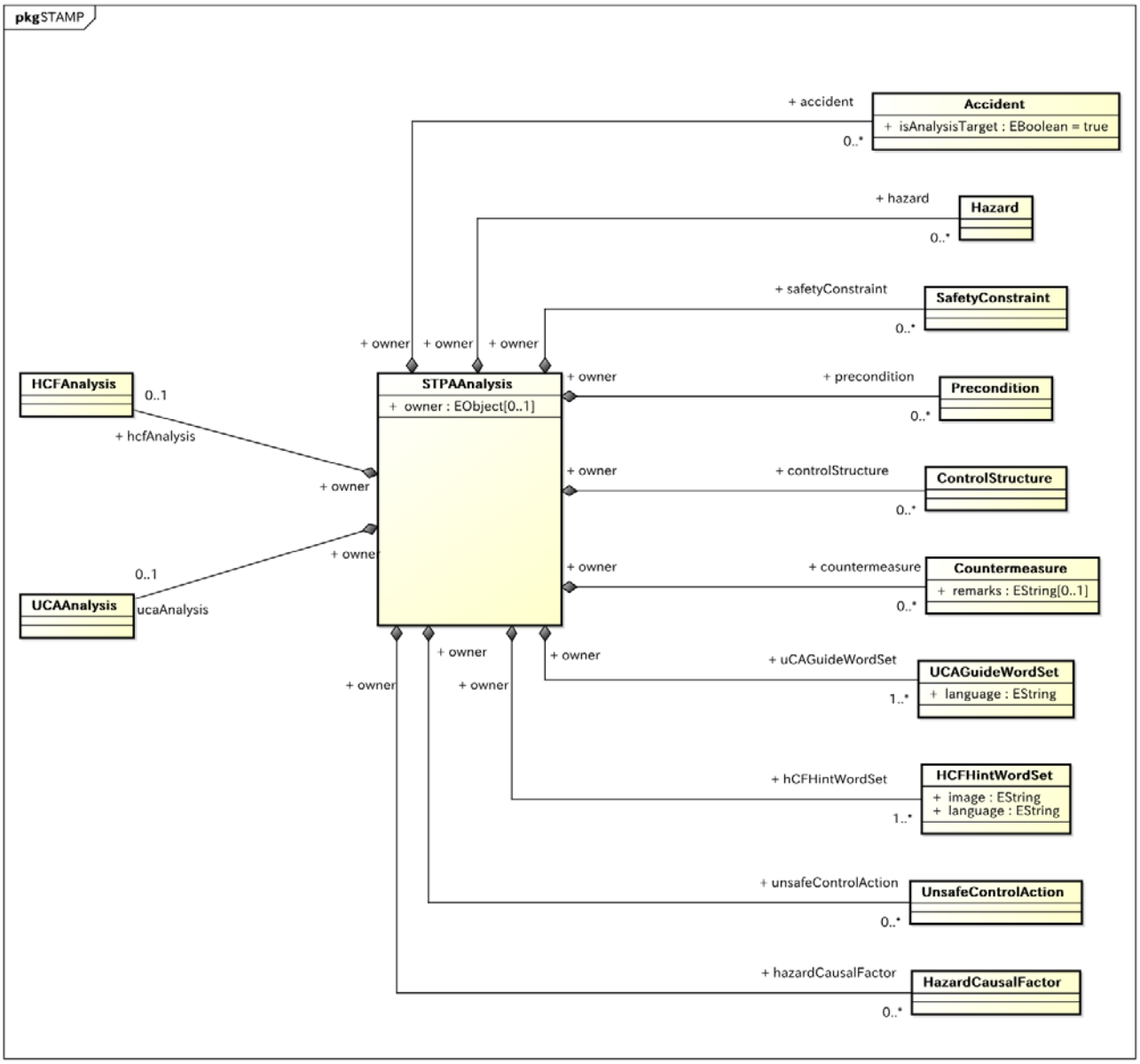
STAMP/STPA のデータモデルの永続化形式は、XMI 2.5.1 相当を採用しており、ドメインモデルと表記法モデルおよび、プロジェクト情報モデルを個別に永続化した XMI ファイルを ZIP で束ねたものになっています。また、これらのモデルと XMI の相互変換には EMF が提供する実装を利用しています。

4.2　　STAMP/STPA メタモデル

このメタモデルは、STAMP/STPA による分析過程および分析結果をモデル化したもので、コンポーネントなどのコントロールストラクチャーや UCA、HCF などの要素を規定します。これらは STPA での分析がどのような概念であるかを示すモデルであるため、図や表での表示に関する情報は含みません。

4.2.1　　基盤モデル





上記は STAMP/STPA の各モデル要素が共通的に利用している基盤モデルのクラス図です。

Element

STAMP/STPA の最上位クラスで、任意の数のノートを設定することができます。

NamedElement

特定の名前空間において、意味を持った名称により他のモデル要素と区別が可能な要素です。

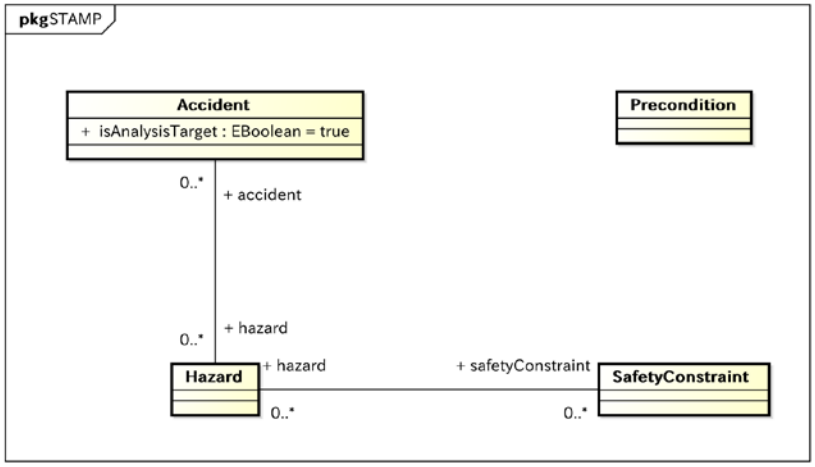
IdentifiedElement

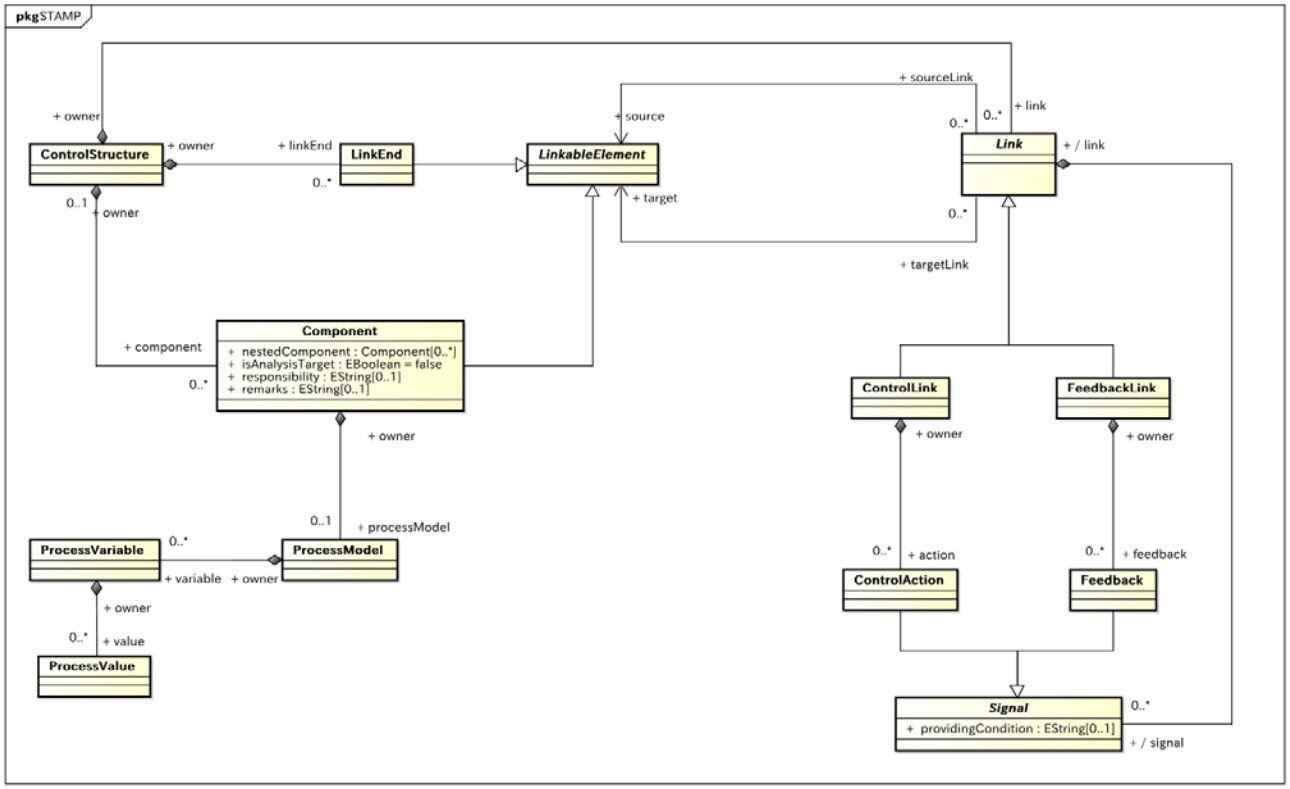
分析モデル全体において、ID により一意に識別可能な要素です。ID は意味を持つとは限らないため、モデルを説明する属性も持ちます。

STPAAnalysis

STAMP/STPA 分析モデルの最上位モデルです。コントロールストラクチャーや UCA 分析、HCF 分析などのモデルのコンテナになっています。

4.2.2 STEP0(準備)のモデル





上記は STEP0(準備)フェーズの範囲で構築するモデル要素のクラス図です。

Accident, Hazard, SafetyConstraint

アクシデントハザード安全制約表が作成するモデル要素です。モデル間の関係はコンポジションではありませんが、他のモデルから利用されていることを寿命とするため、複数のコンポジションの所有者に

共有されているように扱われます。

Precondition
前提条件表が作成するモデル要素です。

ControlStructure
コンポーネント抽出表、および、コントロールストラクチャー図が対象とするモデル構造です。

Component
コンポーネント抽出表における登場人物、および、コントロールストラクチャー図で作図するコンポーネントのドメインモデルです。

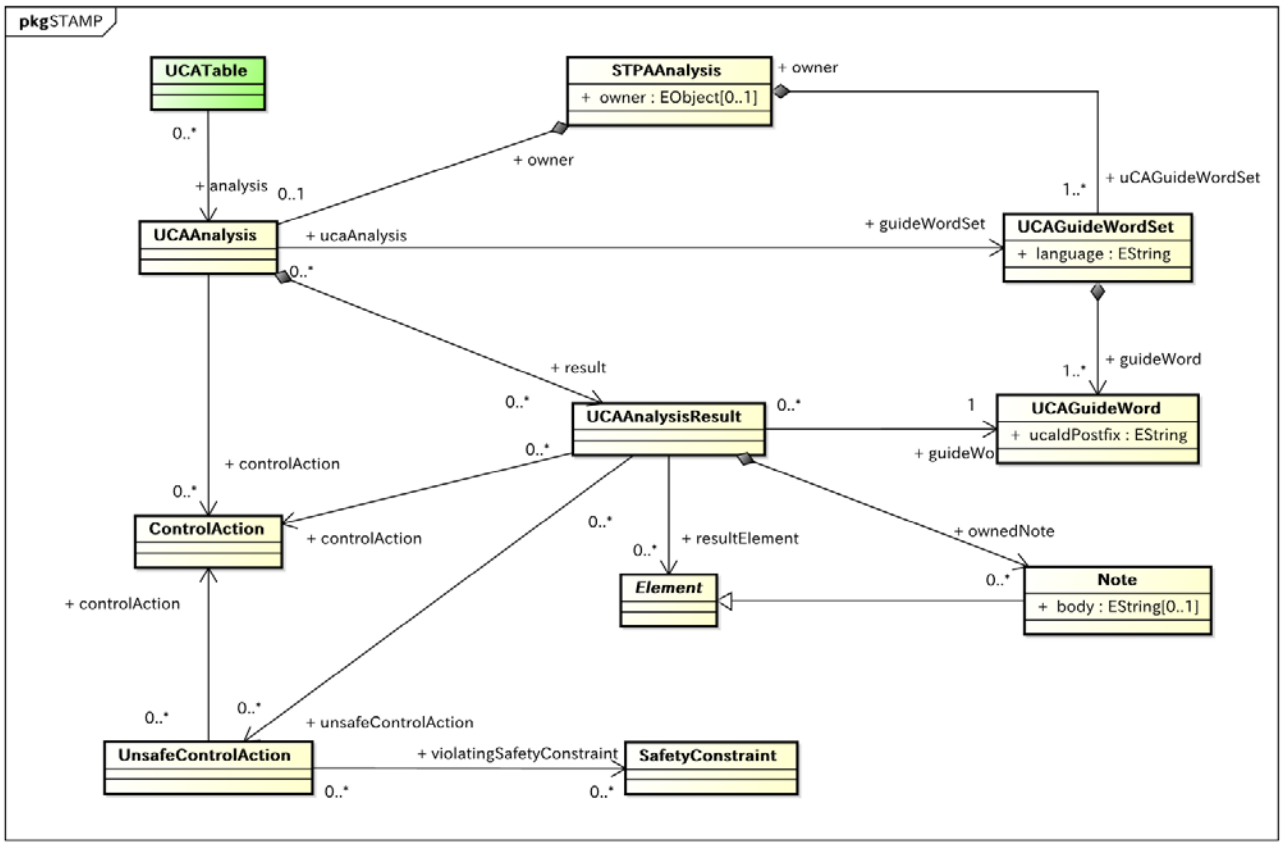
ProcessModel, ProcessVariable, ProcessValue
コンポーネントに設定するプロセスモデルのモデルです。

ControlLink, FeedbackLink
コントロールストラクチャー図で作図するコンポーネント間のリンクのモデルです。リンクとフィードバックのリンクではモデルが異なっており、それぞれ ControlLink と FeedbackLink が作成されます。

ControlAction, Feedback
コントロールストラクチャー図で作図するコンポーネント間のリンクに設定されるコントロールアクションおよびフィードバックのモデルです。ControlAction は ControlLink に、Feedback は FeedbackLink 上にのみ作成することができます。

LinkEnd
始点・終点のモデルです。Component と同様に LinkableElement を特殊化しており、リンクとフィードバックのリンクを接続することができます。

4.2.3　STEP1(UCA 分析)のモデル



STEP1(UCA 分析)フェーズの範囲で構築するモデル要素のクラス図です。

UCAAnalysis

UCA 分析のモデルです。UCA 表を複数作成した場合もひとつのインスタンスが共有されます。

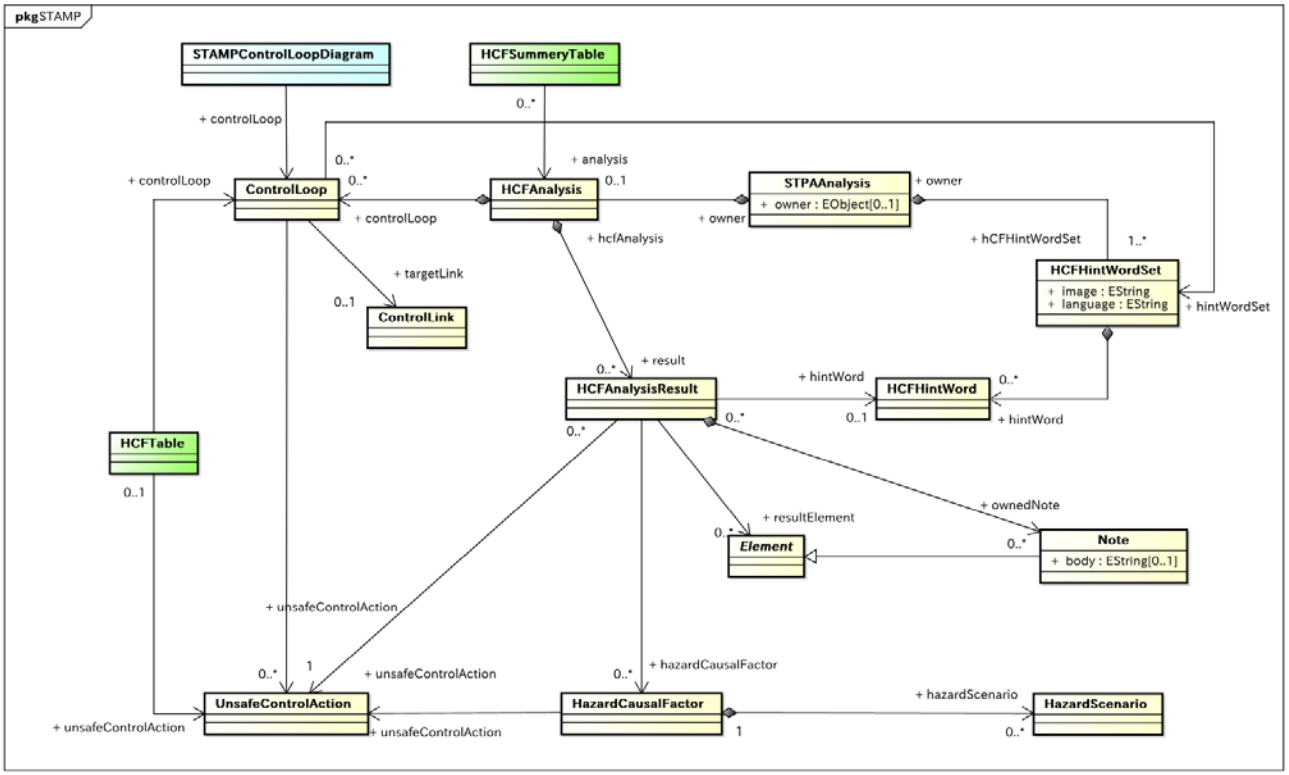
UCAAnalysisResult

UCA 表における、コントロールアクション行とガイドワード列の交点のセルを表現する分析結果のモデルです。UCA と非 UCA の内容である Note はそれぞれ異なる関連で保持されていますが、どちらも resultElement という関連のサブセットであるため、順序はスーパーセット上で表現されています。
なお、このモデルは、同じコントロールアクションを表示するコントロールストラクチャー図を対象とする UCA 表のセルから共有されます。

UnsafeControlAction

UCA 表での分析結果として識別された UCA のモデルです。識別された UCA は分析過程とは独立して利用される可能性を考慮するため、UCAAnalysis のコンポジション構造には含まれません。

4.2.4　STEP2(HCF 分析)のモデル



STEP2(HCF 分析)フェーズの範囲で構築するモデル要素のクラス図です。

HCFAnalysis

HCF 分析のモデルです。HCF 表を複数作成した場合もひとつのインスタンスが共有されます。

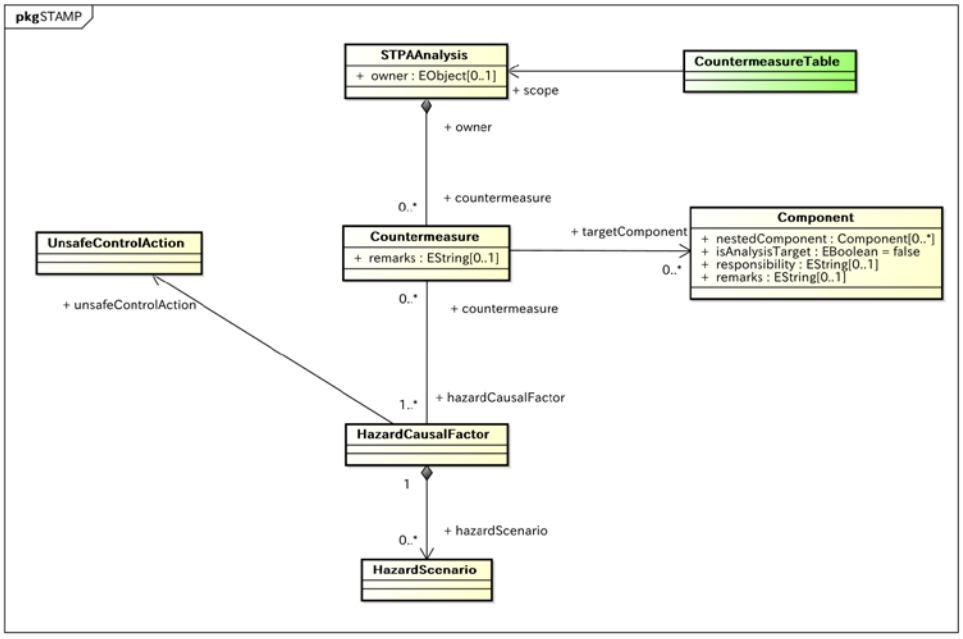
HCFAnalysisResult

HCF 表における、行を表現する分析結果のモデルです。設計上は UCAAnalysisResult と同様に、HCF だけでなく非 HCF の内容を Note として扱うこともできますが、現在は HCF に対して 1 対 1 で作成する方針としています。

ControlLoop

コントロールストラクチャー図のひとつのコントロールリンクを分析するためのモデルで、HCF 表やコントロールループ図を作成すると、対象のリンクに対して作成されます。

4.2.5　対策検討のモデル



対策検討フェーズの範囲で構築するモデル要素のクラス図です。

Countermeasure

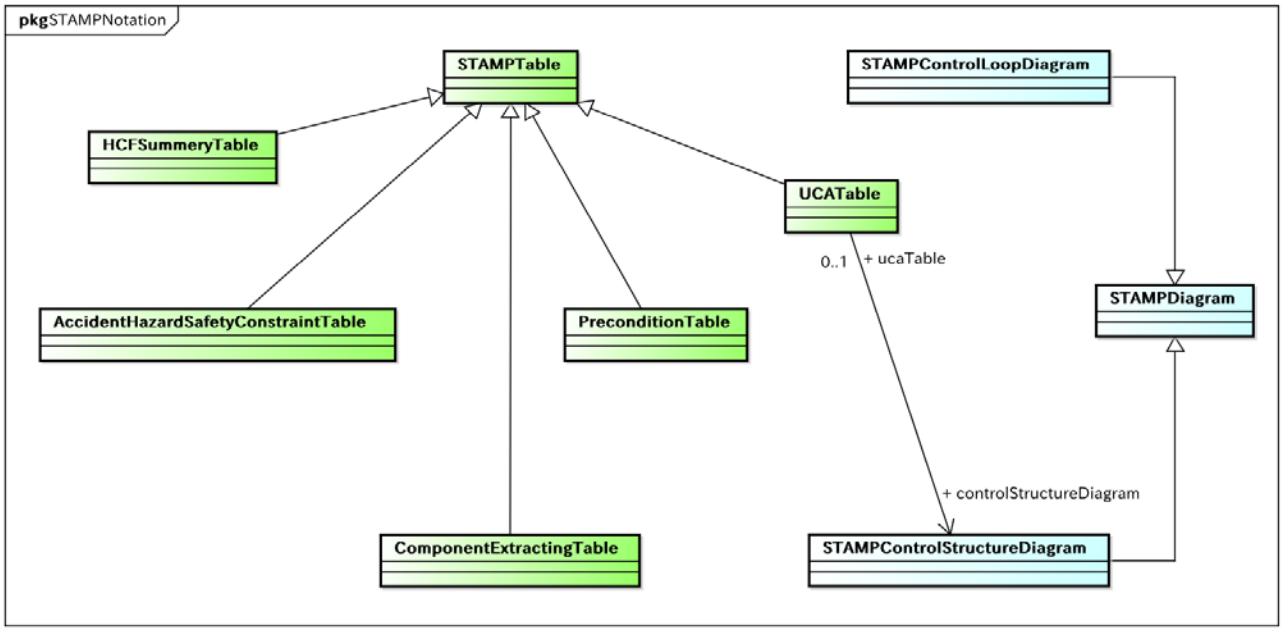
対策表における対策のモデルです。複数の HCF に関連付く場合があり、その場合は対策表の複数の行に対策が表示されます。

4.3　STAMP/STPA 表記法メタモデル

このメタモデルは、STAMP/STPA メタモデルで規定されたモデルを、図や表においてどのように表現するかをモデル化したもので、図や図に含まれる図要素、表などの要素を規定します。

なお、図および図要素の設計は OMG が規定する Diagram Definition (以下、DD) 1.1 に含まれる Diagram Interchange (以下、DI)を踏襲しており、UML 2.5 仕様に含まれる UML DI 仕様と類似したものになっています。

4.3.1　図や表のモデル



STAMP/STPA に登場する図や表のクラス図です。

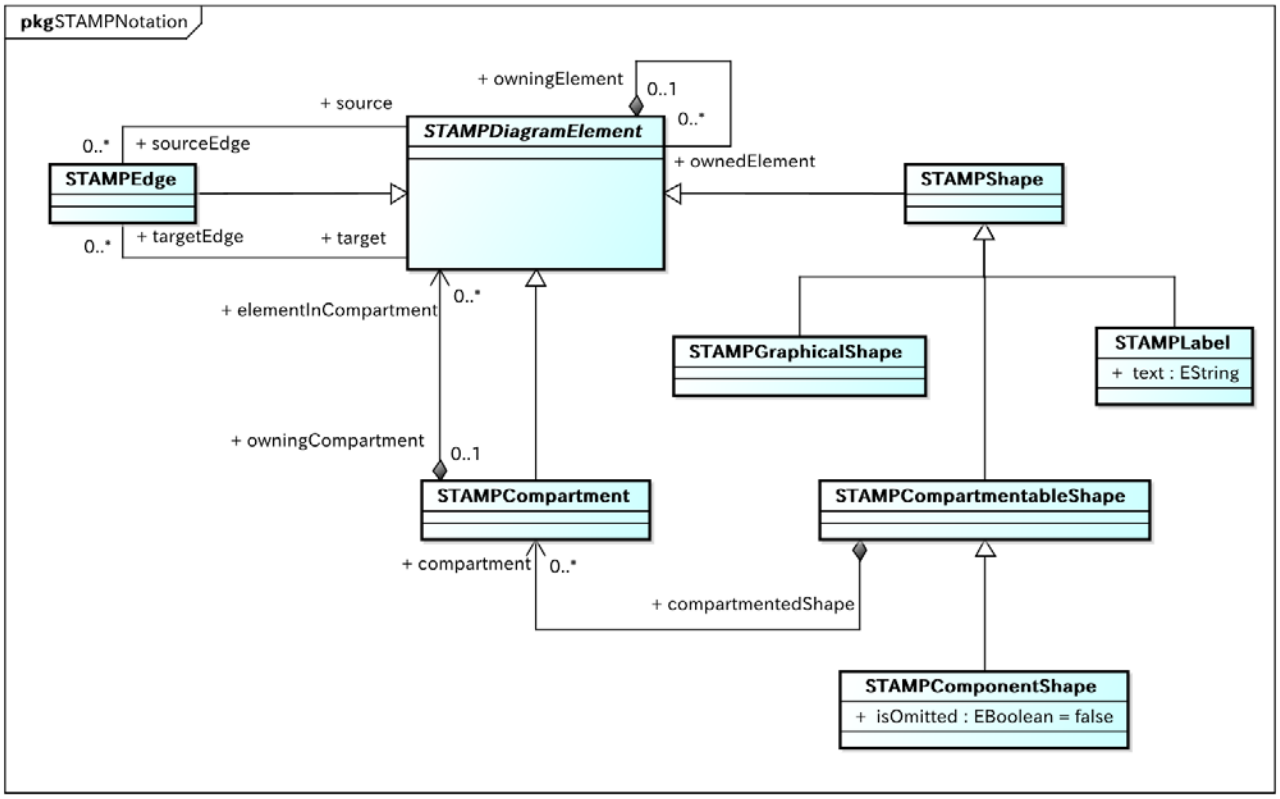
4.3.2　図のモデル

コントロールストラクチャー図(STAMPControlStructureDiagram)とコントロールループ図(STAMPControlLoopDiagram)のモデルが定義されています。

4.3.3　表のモデル

前提条件表(PreconditionTable)とアクシデントハザード安全制約表(AccidentHazardSafetyConstraintTable)、コンポーネント抽出表(ComponentExtractingTable)、UCA表(UCATable)、HCF 表(HCFTable)、対策表(CountermeasureTable)のモデルが定義されています。
表のモデルには表の内容である行と列のグリッド状モデル(表としてのビュー)は含まれず、グリッド状モデルを抽出するための情報(主に抽出範囲を特定するためのスコープ）を保持しています。

4.3.4　コントロールストラクチャー図およびコントロールループ図の図要素モデル



コントロールストラクチャー図およびコントロールループ図に登場する図要素のクラス図です。

STAMPDiagramElement

STAMP の図要素の抽象クラスです。DI の DiagramElement を特殊化しており、DI では派生として定義されていた関連を STAMP 表記法モデルの関連として再定義しています。

STAMPShape

STAMP のノード系図要素です。図要素の表示状態を属性として持つ必要がなく、表示すべき内容をドメインモデルから区別できる場合は、特殊化したメタクラスは用いずに、STAMPShape をそのままインスタンス化して利用します。

STAMPEdge

STAMP のエッジ系図要素です。図要素の表示状態を属性として持つ必要がなく、表示すべき内容をドメインモデルから区別できる場合は、特殊化したメタクラスは用いずに、STAMPEdge をそのままインスタンス化して利用します(STAMP では STAMPEdge を特殊化したメタクラスはありません)。

STAMPLabel

テキストを表示するラベルの図要素です。テキストとして表示する内容は属性として保持しますが、ドメインモデルが更新された場合は、図要素のテキスト属性を更新する必要があります。

STAMPCompartmentableShape, STAMPCompartment

内部に区画を持つことができるノード系図要素と、その区画の図要素です。

STAMPComponentShape

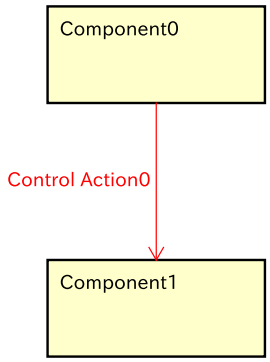
コンポーネント図要素です。コントロールループ図では図外のコンポーネントであることを示す省略表記があり、表示状態を持つ必要があるため、STAMPShape から特殊化したメタクラスを導入しています。

STAMPGraphicalShape

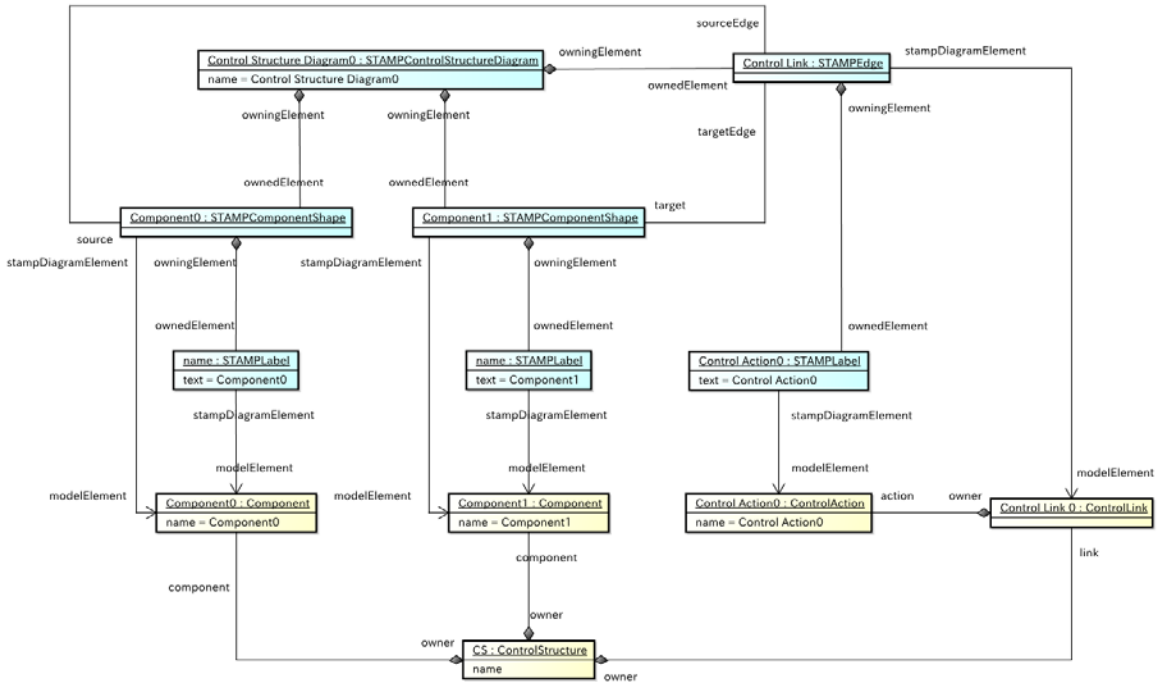
STAMP の図における共通図要素(テキストや四角形、楕円などの図形)です。

4.4 STPA 分析モデルの例

4.4.1 コントロールストラクチャー図の例



次のオブジェクト図は、上記コントロールストラクチャー図のモデルを抜粋したものです。



コンポーネントの図要素

例の図のコンポーネントは枠となる矩形の図要素(STAMPComponentShape)と名前のラベル図要素(STAMPLabel)で構成されており、いずれも Component をモデル要素として参照しています。この例ではプロセスモデル区画はありませんが、表示する場合は、名前のラベルの兄弟要素として STAMPCompartment の複合構造が追加されます。

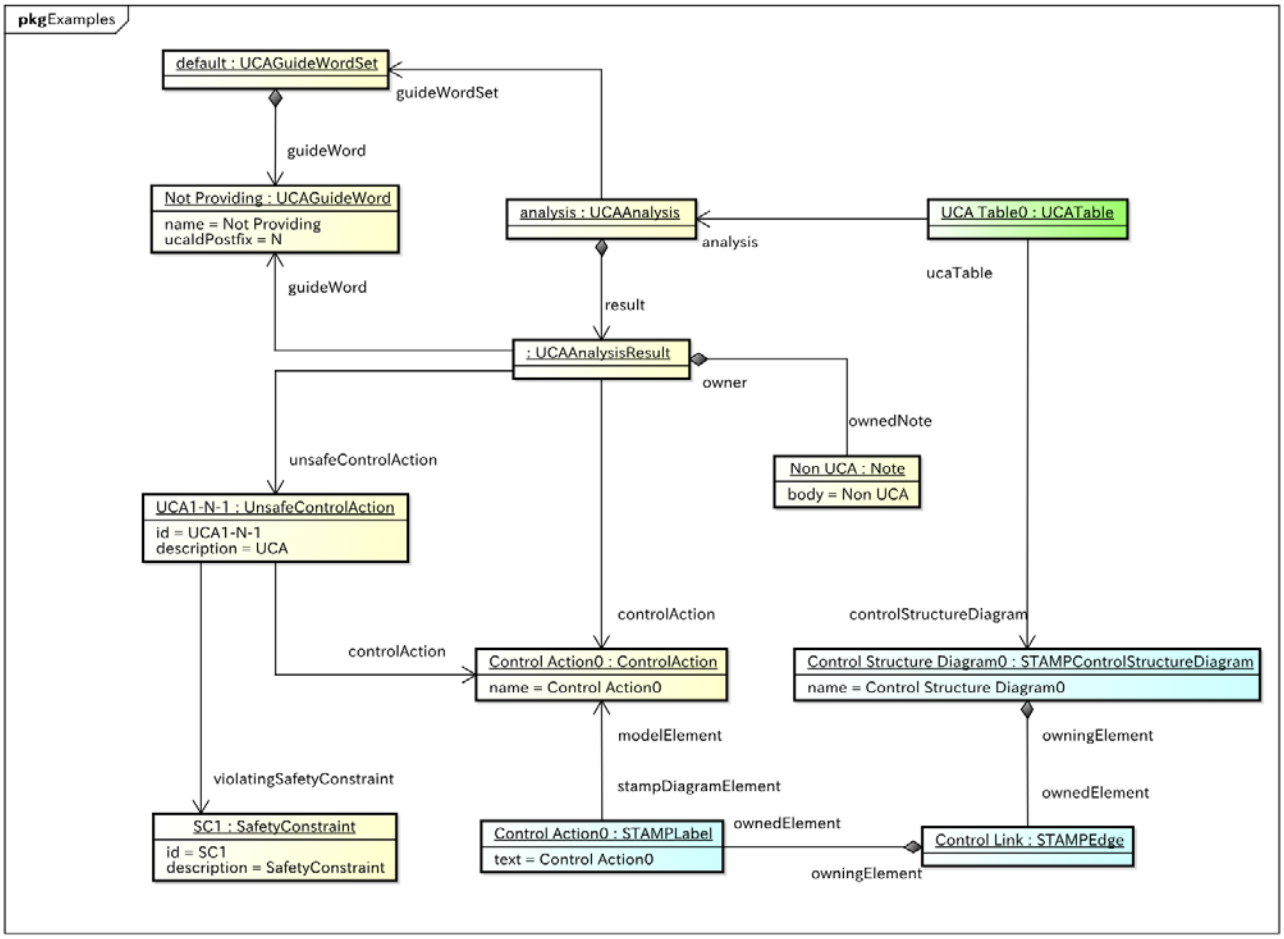
リンクとコントロールアクションの図要素

例の図のリンクは、リンクの図要素(STAMPEdge)からモデル要素として ControlLink を参照し、始端と終端としてそれぞれ STAMPComponentShape を参照します。コントロールアクションはリンク図要素の子要素となっているラベル図要素(STAMPLabel)からモデル要素として ControlAction を参照します。

4.4.2 UCA 表の例

No	CA	From	To	CA Providi...	Not Providing	Providing causes hazard	Too early / Too late	Stop too soon / Applying too long
1	Control Action0	Component 0	Component 1		(UCA1-N-1) UCA [SC1] Non UCA			

次のオブジェクト図は、上記 UCA 表のモデルを抜粋したものです。



UCA 表のモデル

UCA 表(UCATable)には対象となるコントロールストラクチャー図があり、図中に存在するコントロールアクションだけが表の対象となります。しかしながら、UCAAnalysis は共有する形になるため、UCAAnalysisResult も全てが対象となるわけではなく、図中に存在するコントロールアクション(STAMPLabel により参照される ControlAction)に対する分析結果のみが表の対象となります。

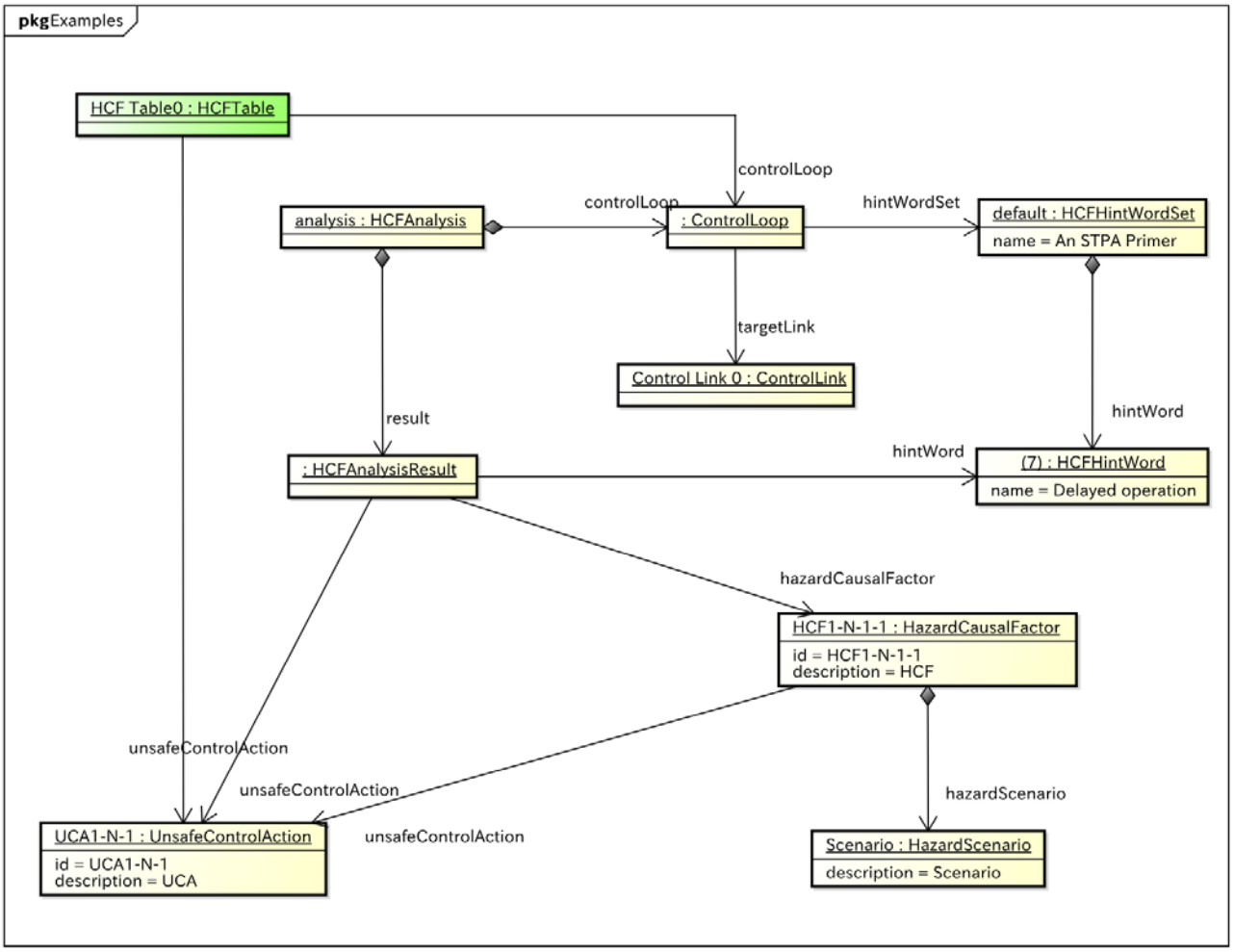
UCA 表の分析結果のモデル

UCAAnalysisResult は対象の ControlAction と UCAGuideWord の組み合わせで一意となるように扱われます。なお、図の例では UCA および非 UCA のテキストが一つずつ登録されていますが、それぞれに複数登録されている場合もあります。

4.4.3 HCF 表の例

ID	HCF	Hint Word	Scenario
HCF1-N-1	HCF	(7) Delayed operation	Scenario

次のオブジェクト図は、上記 HCF 表のモデルを抜粋したものです。



HCF 表のモデル

HCF 表(HCFTable)には対象となる UCA があり、表内で定義される全ての HCF(HazardCausalFactor)が参照する UCA も同じものになります。

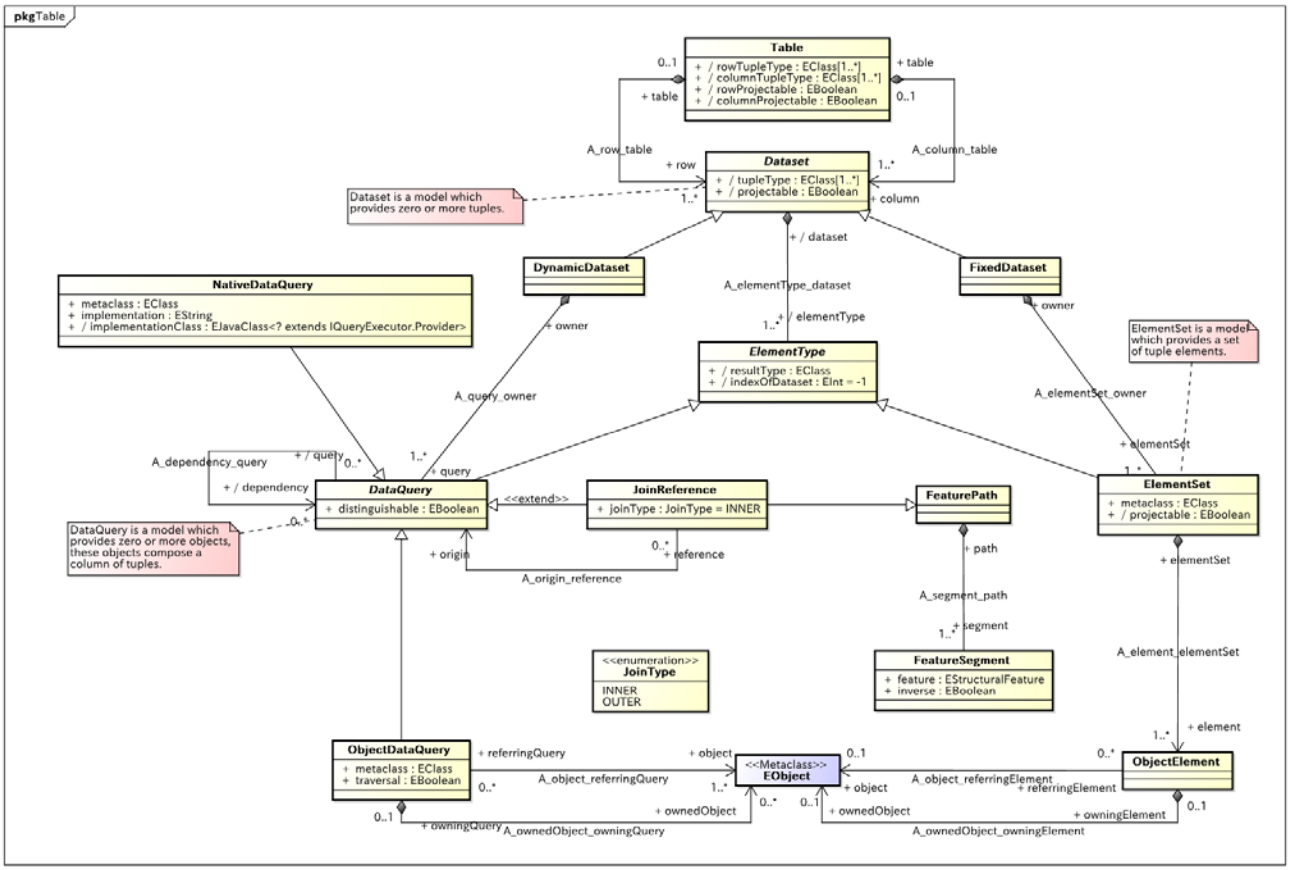
HCF 表の分析結果のモデル

UCA 表の分析結果モデルとよく似た構成ですが、UCA 表とは異なり、HCFAnalysisResult は UnsafeControlAction と HCFHintWord の組み合わせでは一意となるようには扱われず、HCF 表で行が追加された場合は、常に HCFAnalysisResult が新たに追加されます。

4.5 表モデル

表は、行と列の各次元のデータ集合の定義(Dataset)からなる表定義モデルと、データ集合をインスタンス化した要素の組(Tuple)を直積することで得られるグリッドモデルの二種類があります。

4.5.1　表定義モデル



Table

表定義モデルの中心となるモデルで行と列の次元の Dataset を持ちます。それぞれの次元の多重度は[1..*]であるため、ひとつの次元に複数種類のデータセットを組み合わせることもできます。

FixedDataset

データセットの結果をテーブル定義モデル上に記述する固定データセットです。

DynamicDataset

プロジェクト上のモデルをクエリーなどで抽出する動的データセットです。動的データセットの配下にはモデル集合を抽出する DataQuery をひとつ以上持ち、複数のクエリーを指定した場合は、依存関係の上位側から直前の出力結果をパラメータに後続のクエリーを実行して、全てのクエリーの出力を直積した結果をデータセット全体としての出力とします。

ObjectDataQuery

動的データセットで利用可能なクエリーで、任意のモデルもしくはモデルの複合構造に含まれている対象メタクラスのインスタンスをクエリーの出力とします。

JoinReference

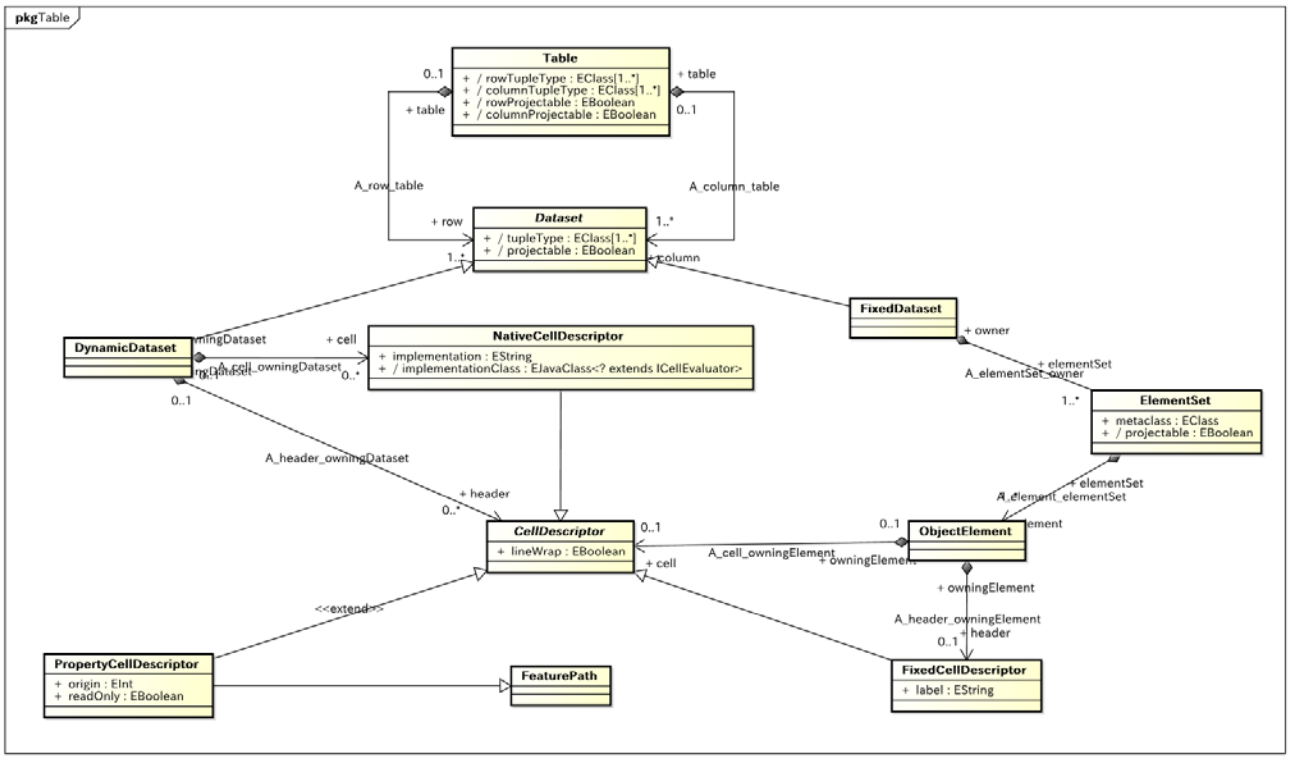
動的データセットで利用可能なクエリーで、指定したクエリーが出力したモデルを起点として参照関係を辿った先のモデルを出力します。

複数のクエリーを指定した場合は直積されるため、このクエリーで定義した参照先が空で出力が空になると Tuple が出力されませんが、JoinType::OUTER を指定した場合は空のモデルを結果として出力することで Tuple が出力されるように構成できます。

NativeDataQuery

動的データセットで利用可能なクエリーで、任意の Java クラスでクエリーを実装します。

4.5.2　セル定義モデル



セル定義モデルは、テーブルの列と行が出力した Tuple を直積した交点であるセル、もしくは、ヘッダ内のセルをどのように表示するかを表現するセル記述子をデータセットに対して定義するものです。

FixedCellDescriptor

固定のラベルをテキストとして出力するセル記述子です。主に列ヘッダなどのラベルを定義します。

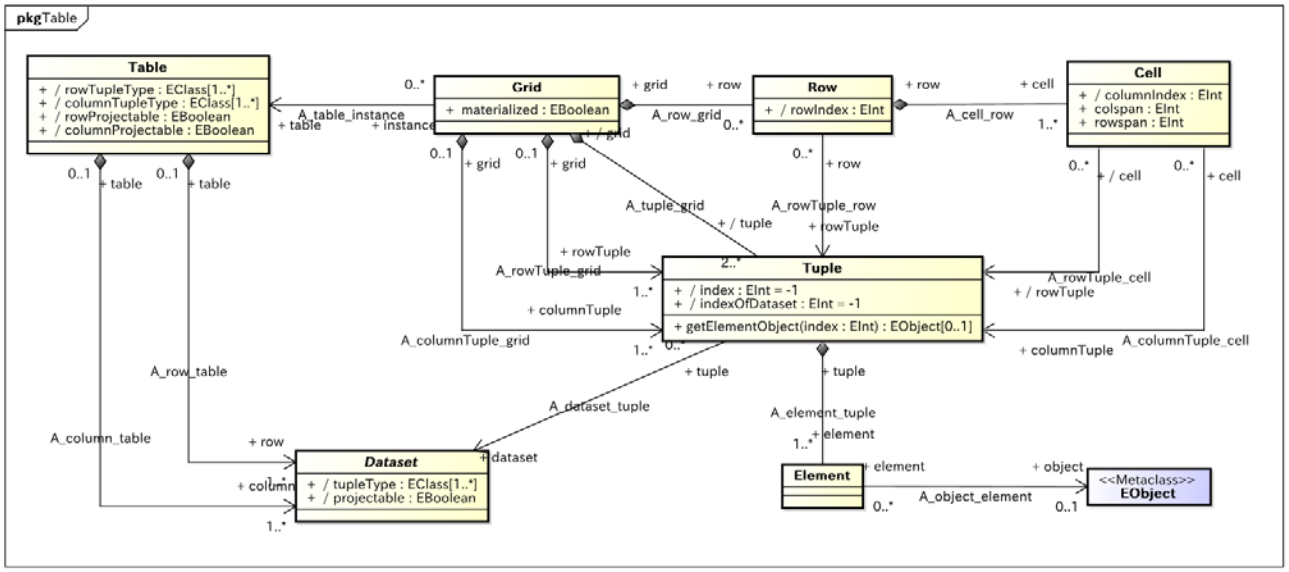
PropertyCellDescriptor

このセル記述子が定義されているデータセットの反対側の次元(行要素のプロパティを表示する場合は、列データセットに定義します)の要素のプロパティをセルとして扱います。

NativeCellDescriptor

行と列の Tuple から、任意の Java クラスを利用して表示するセル記述子です。

4.5.3 グリッドモデル



表定義モデルをインスタンス化したモデルで、STAMP Workbench ではメモリ上にのみ存在しますが、EditingDomain に含めて永続化対象にすることも可能です。

5 ツールプラットフォーム API

5.1 メニュー

メニューはプロパティファイルで定義しており、プラットフォームが提供する標準動作を STAMP Workbench 固有の ManagementWindowPropS.properties を重ね合わせています。プロパティファイル中の定義は、managementview.menubar を最上位とした階層構造になっており、このキーに関連付いた値を空白区切りで分割し、各要素を次の階層のメニューのキーとして扱うことで構築しています。

メニューの各項目は、次の接尾辞を付与したキーに値を関連付けることで定義します。
なお、要素がキーではなく”-“となっている場合は、セパレータとして扱われます。

キーの接尾辞	説明
.image	メニューのアイコン画像の URI です。
.label	メニューのラベル文字列です。 文字列中の”$” はニーモニックとして扱われます。
.action	メニューから実行するアクションです。
.tooltip	メニューのツールチップ文字列です。
.key	メニューをバインドするアクセラレータキーです。

5.1.1　メニューの定義例

次に、プロパティファイルでのメニュー定義の例を示します。managementview.menu.diagram では図メニューに含まれるメニュー項目群を定義しており、そのうちコントロールストラクチャー図の作成のメニュー項目の定義までを記載しています。

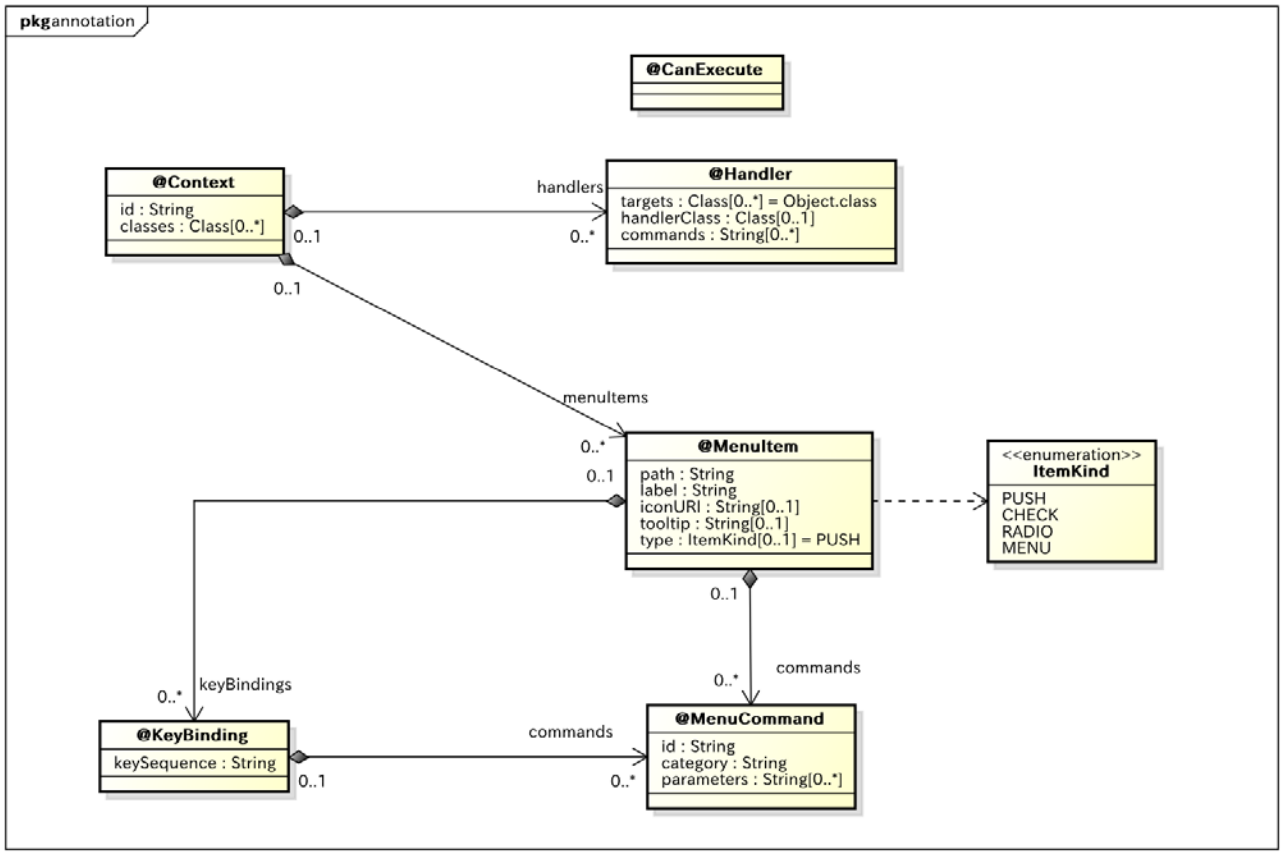
```
managementview.menu.diagram= managementview.menu.diagram.control_structure_diagram ¥
           managementview.menu.diagram.control_loop_diagram - ¥
           managementview.menu.diagram.component_extracting_table ¥
           managementview.menu.diagram.precondition_table ¥
           managementview.menu.diagram.accident_hazard_safety_constraint_table ¥
           managementview.menu.diagram.uca_table ¥
           managementview.menu.diagram.hcf_table ¥
           managementview.menu.diagram.countermeasure_table

managementview.menu.diagram.control_structure_diagram.label=$Control       Structure
Diagram
managementview.menu.diagram.control_structure_diagram.action=net.astah.stpa.stamp.ui
.control.CreateControlStructureDiagramDelegate
managementview.menu.diagram.control_structure_diagram.image=platform:/plugin/net.ast
ah.stpa.stamp.notation.edit/icons/full/obj16/STAMPControlStructureDiagram.png
```

5.2　　ツールバー

ツールバーもメニューと同様にプロパティファイルで定義しており、共通ツールバーの最上位のキーは managementview.menu.edit_toolbar です。その他のプロパティファイルに設定するキーと値の意味は、メニューの項を参照して下さい。

5.3　　コンテキストメニュー

コンテキストメニューは、次のクラス図に示す構造の Java アノテーションで図や表を実装するクラス記述することで定義します。



5.3.1 @Context

個々の図エディタのような、プラットフォーム上の UI 部品などを単位とする文脈を意味する注釈で、メニュー定義の起点になります。STAMP Workbench では図や表のコンテキストメニューの定義に利用しており、図や表の実装クラスがこのアノテーションを定義しています。

5.3.2 @MenuItem

メニューの項目を定義します。定義する内容はテキストやアイコンなど UI 上でのメニューの表現、および、実行対象のコマンド(後述の@MenuCommand)を定義します。

5.3.3 @MenuCommand

メニューやキーバインドから実行可能な抽象的なコマンドを定義します。この定義だけでは実行される処理(後述の@Handler)とは直接的には結びつかず、このメニューに関連するハンドラのうち、メニューを評価しているオブジェクトを対象に実行可能なハンドラが選択されます。

5.3.4 @Handler

何らかのオブジェクトに対応する処理をコマンドに提供します。定義対象のクラスのメソッドに後述の実装メソッドのアノテーションを定義することで、実行可否の評価やコマンド実行時の処理を実装することができます。

5.3.5　実装メソッドのアノテーション(@Execute, @CanExecute, @IsEnabled)

あるオブジェクトに対するコマンドを評価する際に実行するメソッドに定義するアノテーションです。@Execute はハンドラとして実行するメソッドに、@CanExecute はハンドラが対象オブジェクトに対して実行可能であるかを評価する際に、@IsEnabled は実行可能な際に、メニューが有効であるか判定する際に、それぞれ呼び出されます。

5.3.6　アノテーションの定義例

次にハンドラクラスにおけるアノテーションの定義例を示します。このクラスはコンポーネント抽出表からのコンポーネント追加を行う場合の処理を行うクラスで、表の実装の単位で定義する@Context アノテーションから参照されています。

```
@Handler
@MenuItem(commands = @MenuCommand(id = "stpa.CreateShapeCommand",
        category = "edit.table"),
        label = "%ComponentExtractingTable_addComponent_label",
        tooltip = "%ComponentExtractingTable_addComponent_tooltip",
        type = ItemKind.PUSH)
public class AddComponentHandler {
    …
    @Execute
    public void execute(ISelection selection, List<String> params, ActionEvent e) {
        // メニュー実行時の処理を実装する
    }

    @CanExecute
    public boolean canExecute(ISelection selection) {
        // メニューを表示可能か判定する処理を実装する
    }
}
```

5.4　編集処理の拡張

ドメインモデルや表記法モデルの編集は、EMF が提供するコマンドをベースに、単純なコマンドを組み合わせた高水準なコマンドに組み上げることで実装されています。それぞれのコマンドは、編集対象のメタモデルが提供する IEditingDomainItemProvider や IItemDiagramProvider から構築するため、それぞれのメタクラス毎の ItemProviderAdapter の対応する実装をカスタマイズすることにより変更できます。

5.4.1 モデル操作の高水準コマンド

ここではエディタで利用している高水準コマンドを説明します。

CreateChildCommand

親要素となるモデルの配下に子要素となるモデルを追加し、必要な初期化を行います。図要素作成モードから図要素を作成した際のドメインモデルの初期化はこのコマンドを通じて行われます。

InitializeFeatureCommand

モデルのフィーチャー(プロパティ)を適切な値で初期化します。主に名前のデフォルト値として重複しない値を採番するために CreateChildCommand から利用されます。

5.4.2 図要素操作の高水準コマンド

ここではエディタが図要素操作に利用している高水準コマンドを説明します。図要素のレイアウト情報やスタイル情報の持ち方はメタモデル毎に変わる可能性があり、このようなメタモデル毎の違いに依存しない設定方法を提供することを目的としています。

SetLocationCommand, MoveLocationCommand

図要素の位置を設定します。座標情報は、前者は DC::Point で図要素を包含する図の原点からの相対座標として、後者は Notation::Vector で現在位置からの相対座標として指定します。

SetSizeCommand

図要素のサイズを設定します。サイズは DC::Dimension で指定します。

SetBoundsCommand

図要素の位置とサイズを同時に設定します。レイアウト情報は DC::Bounds で図要素を包含する図の原点からの相対座標として指定します。

SetEdgeEndCommand

関係線の接続先および接続位置のレイアウト情報を設定します。

SetEdgeLineCommand

関係線の形状を設定します。

SetStyleCommand

図要素のスタイルを設定します。指定可能なフィーチャーは IItemGraphicalStyleProvider に定義されており、フォントは IItemFontProvider に、色は IItemColorProvider に定義されている URI の形式で指定します。

5.4.3 モデル変更後の同期処理

モデル変更はトランザクションで保護されており、コミット時にはそれぞれのモデルが関心を持つ参照関係に影響する変更があった際に、任意の同期処理をトリガーすることができます。
例えば、ラベル図要素のテキストとして表示するドメインモデルの名前属性に変更があった際に、ラベル図要素の同期処理をトリガーし、ドメインモデルの名前属性を再取得してラベルのテキストに反映するような処理です。

IItemChangeProcessorProvider と LayoutManager

IItemChangeProcessorProvider では IChangeProcessor として汎用的な処理も行えますが、図に関しては専用の ChangeProcessorAdapter があり、図および図要素モデルに変更があった場合は、LayoutManager による図要素の再配置処理を行います。
この LayoutManager の STAMP Workbench の実装である STAMPDiagramLayoutManager では、図内のモデルが変更された場合の共通処理として、Z-Order の再調整を行っています。

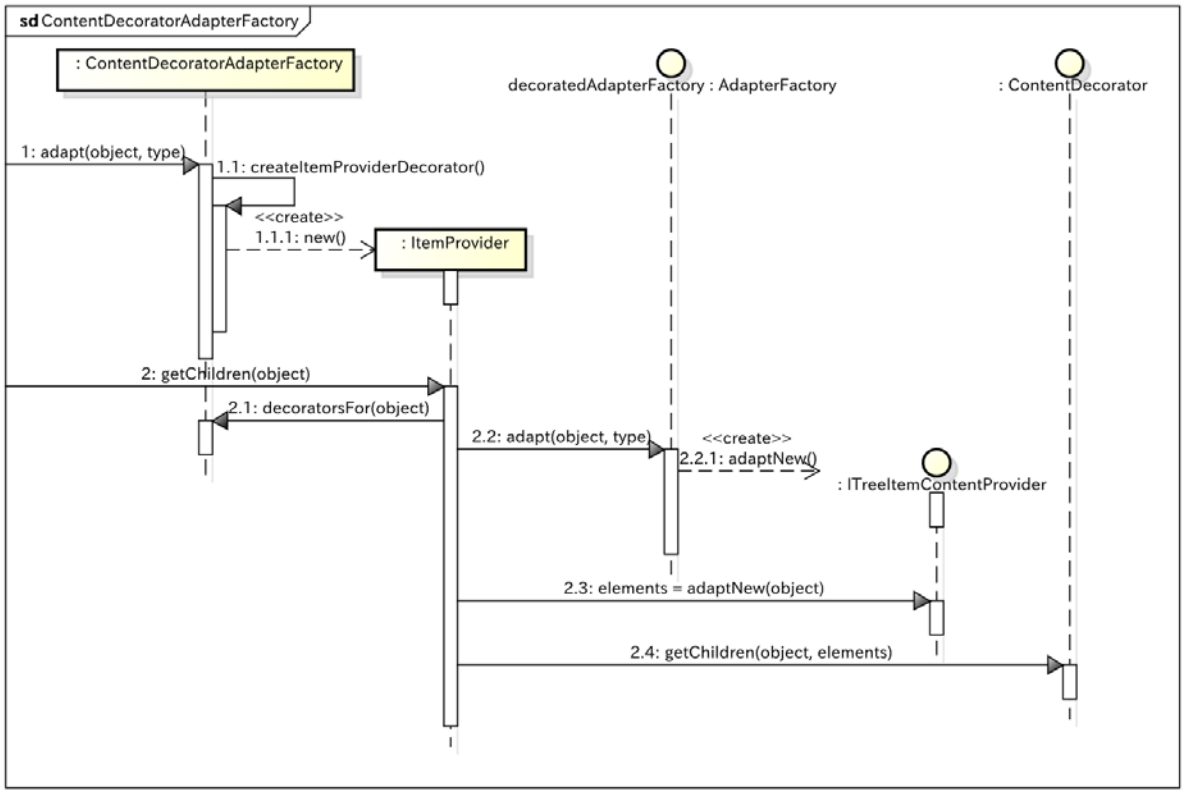
EFeaturePathReactor

あるモデルを起点とした参照関係のパスを定義し、その参照関係に何らかのモデル変更があった場合に何らかのモデル変更処理を実行するための機構です。
例えば、UML のクラスの属性ラベルは名前だけでなく型名も表示する場合がありますが、型名は属性(UML::Property)が参照する型(Property::type)が持つ名前であり、図要素を起点とすると、「図要素→属性→型」という参照関係を辿った先の全てのモデルの変更の影響をトリガーに、ラベルのテキストを更新する可能性がありえます。
EFeaturePathReactor では、このような更新処理、および、トリガーとなる参照関係を plugin.xml で定義することができます。設定の要素などは実行モジュール上に定義されている net.astah.emf.edit.featurePathObservers 拡張ポイント(Eclipse 拡張)を参照して下さい。

5.5 構造ツリーへのモデルの供給

構造ツリーは、ルートとなるプロジェクトモデルを起点に、それぞれのモデルに対応するアダプタを問い合わせて、ITreeItemContentProvider が供給する木構造のモデルを IItemLabelProvider が供給するノードのラベル情報に従って表示しています。
木構造およびノードの表示内容は上記インタフェースを実装するメタクラス毎のプロバイダ実装(ItemProviderAdapter を拡張したクラス)のメソッド(例えば、アイコンは getImage(Object) が返したものが表示されます)をオーバーライドすることでカスタマイズできますが、このようなカスタマイズはメタクラス毎に普遍的なものであるため、複数種類のツリー(例えば、構造ツリー以外にもモデル選択ツリーなどもありえます)に対応可能とするために、plugin.xml に事前定義されたルールで改変(フィルタリングやソートなど)したものを表示することが可能です。

以下に構造ツリーからモデルの木構造を取得する際のシーケンスを示します。



図上の decoratedAdapterFactory と ITreeItemContentProvider は EMF が提供もしくはコード生成する標準の実装で、この実装が返した結果を ContentDecorator が修飾することで本来のモデル構造とは異なった表示を行うことができます。

5.5.1 net.astah.emf.edit.contentDecorators 拡張ポイント

メタモデル毎の修飾ルールは、net.astah.emf.edit.contentDecorators から登録することができます。設定の要素などは実行モジュール上に定義されている当該拡張ポイント(Eclipse 拡張)を参照して下さい。

5.5.2 標準提供の修飾ルール

ContentDecorator の実装である修飾ルールは、標準で次のものが提供されています。

MoveClassRule

指定された参照関係の始端が親、終端が子となるように ITreeItemContentProvider が返す親子関係を修飾します。

FilterRule

対象のメタクラスに対して、条件に ITreeItemContentProvider が返す子要素をフィルタリングします。

SortRule

対象のメタクラスに対して、ITreeItemContentProvider が返す子要素を、指定された比較関数(Comparator)を使用して順序付けします。

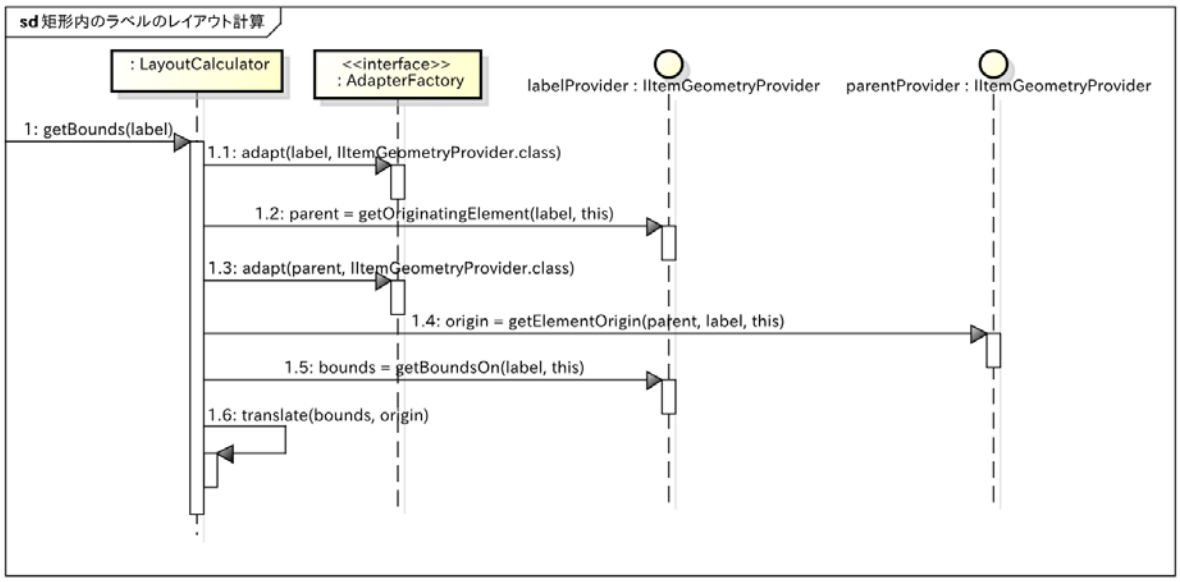
5.5.3　修飾ルールの定義例

STAMP Workbench では次のような実装を拡張ポイントに登録することで、構造ツリーに表示するモデルをフィルタリングしています。

```
public class STAMPTreeDecoratorFactory extends BasicContentDecoratorFactoryImpl {
    …
    @Override
    protected void addContentDecorators(final Registry registry) {
        super.addContentDecorators(registry);

        // STPAAnalysis 直下の Component 以外のモデルをフィルタする
        registry.addDecorator(FilterRule.excludeKindOf(
                STAMPPackage.Literals.STPA_ANALYSIS,
                asList(STAMPPackage.Literals.IDENTIFIED_ELEMENT,
                        STAMPPackage.Literals.UCA_GUIDE_WORD_SET,
                        STAMPPackage.Literals.HCF_HINT_WORD_SET,
                        STAMPPackage.Literals.CONTROL_STRUCTURE,
                        STAMPPackage.Literals.UCA_ANALYSIS,
                        STAMPPackage.Literals.HCF_ANALYSIS)),
                Priority.FILTER.value());
    }
}
```

5.6　図要素のレイアウト

それぞれの図要素は、メタモデル毎にカスタマイズ可能なプロバイダインタフェースが返す結果を組み合わせて求められる。次のシーケンスでは、STAMP のコンポーネントのように、矩形の中に名前のラベルがある場合のバウンディングボックスの計算処理を説明します。



1. ラベル図要素のプロバイダ実装に、座標の基準となる図要素を問い合わせる(1.2)
2. 基準となる親図要素のプロバイダ実装に、子図要素の基準位置を必要に応じて再帰的に問い合わせる(1.4)
3. ラベル図要素のプロバイダ実装に、親図要素内からの相対でのバウンディングボックスを問い合わせる(1.5)
4. (1.5)のバウンディングボックスを(1.4)の座標系に移動することで絶対座標を得る(1.6)

5.6.1 親図要素内での図要素の配置

先ほどのシーケンスの 1.4: getElementOrigin()では、ラベル図要素の基準座標を返す必要がありますが、テキストを中央揃えさせるような場合や、改行を含むテキストを複数並べる場合などは、配置を適切に計算する必要があります。

この場合は、レイアウト計算機構を利用することでレイアウト情報の定義により、計算を自動化することができます。定義例は STAMPCompartmentableShapeItemProvider などの実装コードを参照して下さい。

5.6.2 関連する図要素に依存したレイアウト

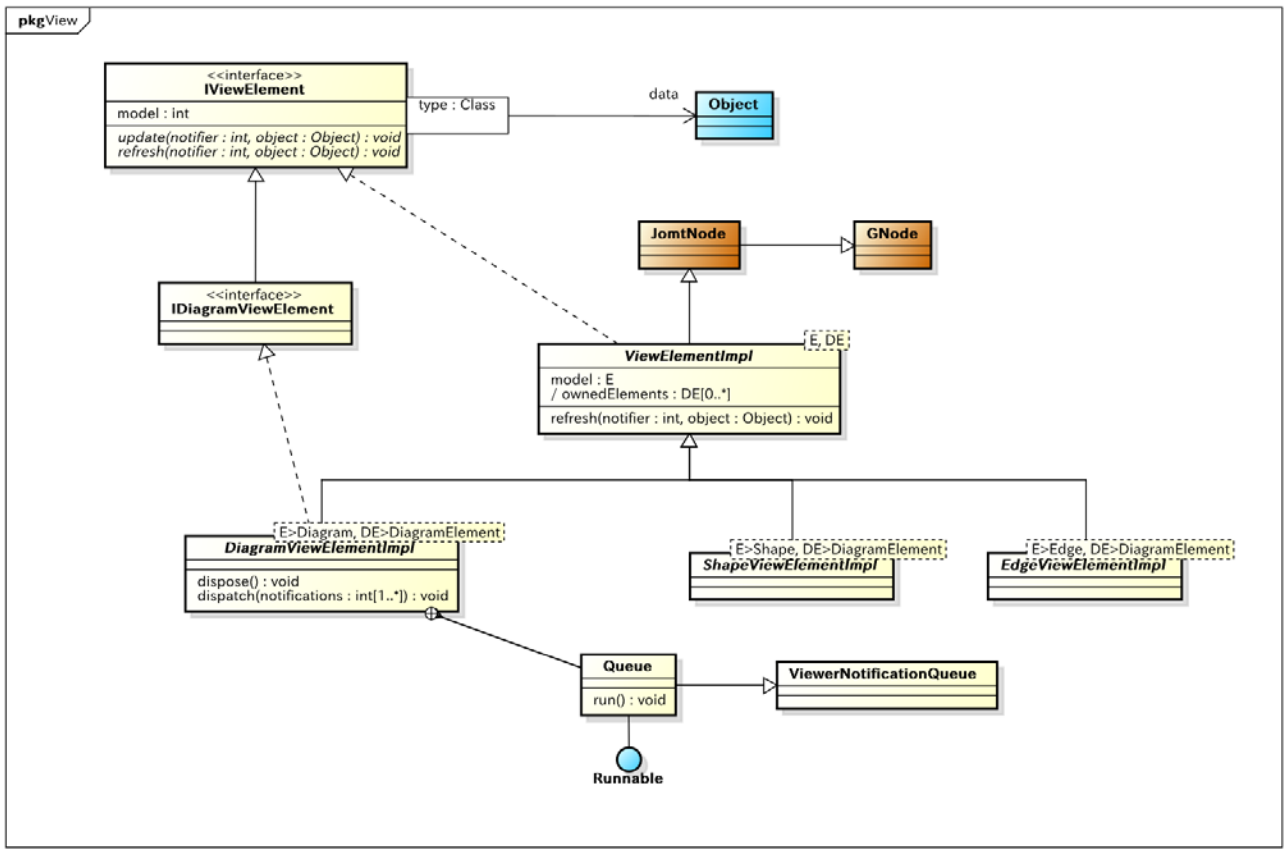
先ほどのシーケンスでは、比較的単純な場合を説明しましたが、ネストを持つ親図要素のサイズ計算や、関係線のラベルの場合は他の図要素の座標やサイズ、バウンディングボックスを利用して計算することになります。

このため IItemGeometryProvider には、バウンディングボックスの取得だけではなく、バウンディングボックスの座標とサイズの各部分を取得する getLocationOn()と getBoundsSize()が定義されています。

他の図要素のレイアウトに依存する場合は、循環する依存関係による無限再帰を防ぐために、依存する図要素のレイアウト計算に自身のレイアウト情報が必要とならないようにレイアウトロジックを構成する必要があります。

5.7　　ビューエレメント

ビューエレメントでは、表記法の表現に従った GNode の描画ツリーを構築し、対応する図要素のプロバイダに計算させたバウンディングボックスなどのレイアウト情報を反映させます。



5.7.1　　ビューエレメントの基底クラス群

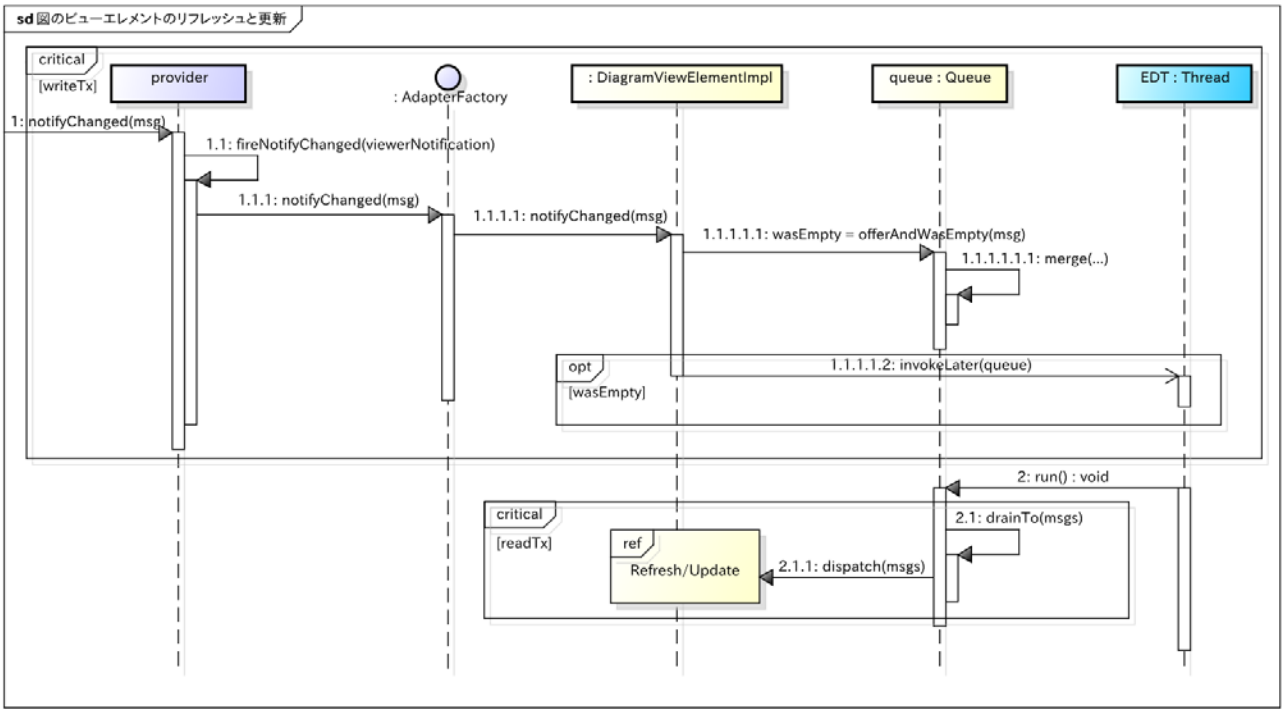
ViewElementImpl

ビューエレメント実装の基底クラスです。ビューエレメントは対応する表記法モデルを参照しており、対応する表記法モデルの複合構造に変化があった場合は、その変更通知からビューエレメントの階層構造に反映させます。

また、表記法モデルのレイアウトに変更があった際に影響を受けるビューエレメントを特定し、最小限の範囲の描画ツリーを更新するためのレイアウト計算ロジックも持ちます。

DiagramViewElementImpl

表記法モデルの Diagram に対応するビューエレメント実装です。図エディタの実装とのインタフェースとなるメソッドのほか、トランザクション完了時の変更通知を受け取って図内の各ビューエレメントに処理させる起点の役割も持ちます。



上記はトランザクション完了処理からのビューエレメントの更新シーケンスで、トランザクション中に検出したモデルの変更通知はキューイングされた上で、EDT からビューエレメントのツリーの更新処理を実行します。この処理は読み取り専用トランザクションで実行されるため、モデルを変更したトランザクションが完了するまでは実行されることはありません。

ShapeViewElementImpl

表記法モデルの Shape に対応するビューエレメント実装です。レイアウト計算処理では、Shape に対して計算させたバウンディングボックスを描画ツリーに反映させるためのメソッドを持ちます。

EdgeViewElementImpl

表記法モデルの Edge に対応するビューエレメント実装です。レイアウト計算処理では、Edge に対して計算させた通過点を描画ツリーに反映させるためのメソッドを持ちます。

5.7.2 ドメインモデルの変更によるビューエレメントの更新

ビューエレメントの更新が必要な状況は、対応する表記法モデルの属性が変更されたことにより発生する直接的な変更のほか、表記法モデルがさらに対象にしているドメインモデルの変更により発生する場合もあります。

このような依存関係は、多段の間接的な参照関係の影響を受ける場合があり、その場合に対応できるように拡張ポイントにて影響を受ける参照関係を定義することができます。詳しくは実行モジュール上に定義されている net.astah.emf.edit.featurePathObservers 拡張ポイント(Eclipse 拡張)を参照して下さい。

5.8 プロパティビュー

モデルに応じた内容をプロパティビューに表示するには、ModelPaneManager を拡張したクラスで、メタモデルに対応した ModelTabGroup を構築する Switch クラスを追加することで行います。
STAMP Workbench では STAMPModelPaneManager で登録している STPADiagramTabGroupSwitch および STAMPModelTabGroupSwitch がそれぞれ表記法モデルとドメインモデルに対応しています。

5.8.1 プロパティビューを構成するクラス

ModelTabGroup

選択されたモデルのプロパティビューの内容として表示する、複数タブで構成されるタブグループを実装します。具体的な実装としては、一般的にタブグループを構成するタブの登録処理のみになります。

ModelTab

タブグループを構成するタブの実装です。モデルの編集を行うタブの場合は PropertySourceTab を拡張することを想定しており、このクラスでは EMF がコード生成する IItemPropertySource を利用したプロパティの編集をサポートしています。

5.9 ID の自動採番

一定のルールに従った ID の自動採番を実現するために、トランザクション完了処理の一環として ID に影響を及ぼす変更を検出して再採番を行うメカニズムを提供しています。

5.9.1 メタモデルごとの定義

IdReorganizer

ID を書式化するための書式定義オブジェクトを集約して管理するオブジェクトで、モデルの変更通知を処理して ID の再採番を行うコマンドを構築して返します。

IdReorganizerListener

IdReorganizer を構築してトランザクション完了処理として再採番処理を起動します。このクラスは STAMP Workbench で実装しており、STAMP/STPA の採番ルールを IdReorganizer に定義しています。

5.9.2 採番ルールの定義

IdSequencer

ID を構成する値を生成する採番オブジェクトです。採番された値は、接頭辞や他のオブジェクトから参照される形で、ID の一部として書式化されます。

IdSequencerFormat

メタクラス単位の書式定義オブジェクトです。採番するフォーマットおよびパラメータとして参照する IdSequencer が定義されたオブジェクトへの参照パスを指定することで定義します。

ReferenceIndexSequencer

参照パスを探索した際の出現順を連番として使用する採番オブジェクトです。この採番オブジェクトを使用するルールでは、指定した参照パスに影響を検出した場合に再採番されます。

PatternSequencer

オブジェクトの ID の現在地に正規表現パターンをマッチングし、先頭グループの文字列を他の採番オブジェクトが出力した値に置き換えた文字列を採番結果として返します。

AttributeSequencer

IdSequencer の実装のひとつで、オブジェクトの特定の属性の値をそのまま採番に使用します。この採番オブジェクトは、対象のオブジェクト自体の自動採ではなく、他のオブジェクト用のルールから参照するためだけに使用します。

以上