Nirvana ver.4.1.0

プラグイン開発者マニュアル

（初版）

初版 2011 年 6 月 8 日

株式会社ニルソフトウェア

このソフトウェアの仕様および関連ドキュメントの記載事項は，将来予告なしに変更する場合があります．

このソフトウェアおよび関連ドキュメント(以下，本ソフトウェアと記します)で使用している会社等の組織・団体，人物，製品等の名称は架空のものであり，特に明記している場合を除き，実在の団体名，個人名，製品名等とは一切関係ありません．

本ソフトウェアは，お客様が使用許諾契約書の内容に同意いただける場合にのみ使用することができます．

本ソフトウェアのご使用にあたり，お客様は本ソフトウェアに含まれる使用許諾契約書に同意していただく必要があります．

目次

1. はじめに

1.1. 目的

本書は, Nirvana のプラグインを作成するにあたり必要な事柄についてまとめたものです.

1.2. APIリファレンス

プラグインの API の詳細については, 同梱の Plugin API リファレンス(HTML)をご覧下さい.

1.3. 用語

本文書で使用する用語を表 1.3. に示します.

表 1.3. 用語

用語	意味
Nirvana	株式会社ニルソフトウェア社製, ソフトウェア開発の分析・設計を支援するモデリングツール.
Nirvana プラグイン	Nirvana の機能を拡張するモジュール.
Nirvana プラグイン API	Nirvana プラグインを作成する際に使用する公開 API.
プラグイン	特に断りがない場合は Nirvana プラグイン.
JAR ファイル	Java Archive ファイル.
Nirvana 本体の JAR ファイル	Nirvana のインストール先フォルダに存在する nirvana.JAR ファイル.
プラグインパッケージ名	プラグインで使用するパッケージ名. 命名規則は 4.2.1 項を参照して下さい.
表記法	UML などの表記法.
図	表記法における図の種類, または, 個々の図を指す.
図の描画要素	図に配置可能な描画要素.
シンボル	図の描画要素の内, 単一で図に配置可能なもの.
関係線	図の描画要素の内, 単一で図に配置不可能であり, シンボルに接続する形で図に配置可能なもの.

1.4. 構成

本文書の構成について説明します. []内は対応する章を指します.

(1) Nirvana プラグインとは [2 章]
Nirvana プラグインの概要, プラグインで拡張可能な機能, 動作環境, Nirvana 本体への組み込み方法といった, 基本的な事柄について説明します.

(2) 準備 [3 章]
Nirvana プラグインの作成に必要な前提とします. 知識, 開発環境について説明します.

(3) Nirvana プラグインの作成 [4 章]
プラグイン API の概要, リソースファイル, 各機能の具体的な実装方法について説明します. プラグインの作成に必要な要点の説明までとし, 各クラスやインターフェースの詳細までは説明しません. 各クラスやインターフェースの詳細については Nirvana プラグイン API リファレンスを参照して下さい.

(4) Nirvana プラグイン API リファレンス [5 章]
プラグイン API を構成するパッケージと説明を示します. 各パッケージに含まれる各クラスやインターフェースの詳細については添付資料(1)の HTML を参照して下さい.

2. Nirvanaプラグインとは

本章では，Nirvana プラグインの概要，プラグインで拡張可能な機能，動作環境，Nirvana 本体への組み込み方法といった，基本的な事柄について説明します．

2.1. 概要

Nirvana プラグインとは，Nirvana ver.4.0 から搭載されたプラグイン機能を用いて，Nirvana で扱える表記法を追加拡張するためのモジュールのことです．Nirvana プラグインは 1 つの JAR ファイルの形態を取ります．

2.2. プラグインで拡張可能な機能

Nirvana プラグインでは以下の拡張が可能です．(1)と(2)はプラグインとして実装や定義が必須です．(3)～(8)は必須ではなく拡張しなくても良いものです．

(1) 表記法の追加
Nirvana プラグイン 1 つに対して 1 つの表記法を追加できます．Nirvana プロジェクトデータの直下に表記法名のフォルダが作成され，その表記法の図を作成すると表記法フォルダ以下に作成されます．

(2) 図の追加
1 つの表記法に対して 1 つの種類の図を追加できます．図の追加に付随して以下の機能拡張ができます．

(a) 図の新規作成メニューの追加
追加するメニューはメニューツリー上では以下の構成になります．
(i) メニュー → ファイル → 新規作成 → ダイアグラム →［追加した表記法名］→［追加した図名］

(b) 図の描画要素の追加
1つの図に対して，複数の種類のシンボルと関係線を定義できます．図の描画要素に関する機能として作図のための編集機能以外に以下の機能を拡張します．
・各描画要素の初期設定色
・Nirvana 本体の画面構成において右下に表示される，選択した描画要素のプロパティービューの拡張機能
・シンボルメニュー，関係メニュー項目やツールバー
追加するメニューはメニューツリー上では以下の構成になります．
(i) メニュー → シンボル →［追加したシンボル名］
(ii) メニュー → 関係 →［追加した関係線名］
(iii) ツールバー →［追加したシンボルや関係線のアイコン］
・シンボルから，他の図へのリンク，または，URI を用いた外部へのリンク
シンボルから他の図へのリンク（以下，図リンクと記す）は，シンボルのポップアップメニューに「リンク」が追加され，そのシンボルから他の図へリンクを追加，変更，削除，リンク先への移動が行えるようになります．
シンボルから URI を用いた外部へのリンク（以下，外部リンクと記す）は，ユーザがシンボルのポップアップメニューに「リンク」とその下位に「外部リンク」が追加され，そのシンボルから，URI やファイルのプロジェクトからの相対パスを指定の追加，変更，削除，その URI やファイルに対応するアプリケーションの起動が行えるようになります．

(c) 図の保存, 読込機能
追加した図を XML 形式で保存, 読込する機能を実装します.

(3) 追加した図を開いている場合のみ使用する処理の追加
追加した図を開いている場合のみ表示されるメニューを追加できます. 追加するメニューはメニューツリー上では以下の構成になります.
(a) メニュー → 編集 → [追加したメニュー項目名]
(b) メニュー → 図 → [追加したメニュー項目名]

(4) シートレイアウトのグリッドサイズ設定機能の変更
シートレイアウトのグリッドサイズ設定機能を変更できます. 以下のメニューで辿り, 表示されるダイアログ内の「グリッドサイズ」タグの内容です.
(a) メニュー → 図 → シートレイアウト

(5) インポート機能の追加
XMI からのインポート機能を追加できます. 追加するメニューはメニューツリー上では以下の構成になります.
(a) メニュー → ファイル → インポート → From XMI → [追加したメニュー項目名]

(6) エクスポート機能の追加
エクスポート機能を追加できます. 追加するメニューはメニューツリー上では以下の構成になります. (a)はメニュー階層「XMI」にメニュー項目を追加する場合. (b)はメニュー階層の「エクスポート」にメニュー項目を追加する場合です.
(a) メニュー → ファイル → エクスポート → XMI → [追加したメニュー項目名]
(b) メニュー → ファイル → エクスポート → [追加したメニュー項目名]

(7) Nirvana の環境設定にプラグイン向けの設定項目の追加
Nirvana の環境設定にプラグイン向けの設定項目を追加できます.

(8) プロジェクトのプロパティにプロジェクトタイプの追加, 参照プロジェクト設定項目の追加
プロジェクトのプロパティ情報でプロジェクトタイプ(プロジェクトの種類)の選択肢を追加できます. これは個々のプロジェクトに対してプロジェクトの種類を指定して何かを行うプラグインを作成する場合などの使用を想定しています.

また, 参照プロジェクト設定項目の追加もできます. 参照プロジェクトとは他のプロジェクトを参照する情報の事です. 複数のプロジェクトを組み合わせて何かを行うプラグインを作成する場合などの使用を想定しています.

(9) プロジェクトのプロパティにプラグイン固有の任意データを追加
プロジェクトのプロパティ情報にプラグイン固有の任意データを追加できます. この情報は Nirvana のプロジェクトデータとして永続化されます. また, その情報の編集画面をプロジェクトのプロパティダイアログに追加できます.

上記に記していない箇所(Nirvana 本体左下のあるオーバービューなど)については, プラグインによる拡張はできません.

2.3. 動作環境, Nirvana本体への組み込み方法

動作環境は, Nirvana 本体の動作環境に準じます.
Nirvana ver.4.0 向けの Nirvana プラグインの動作環境は, JRE(Java Runtime Environment) 6 が動作する下記の環境です.
・Microsoft Windows 7, Vista SP2, XP Professional SP3
・Linux (Ubuntu 10.04LTS にて確認)

Nirvana 本体への組み込みは, Nirvana 本体の plugins フォルダにプラグインの JAR ファイルを追加します. それにより, 起動時にそのプラグインが読込まれ, そのプラグインの拡張が Nirvana に反映されます.

3. 準備

本章では，Nirvana プラグインの作成に必要な前提とする知識，開発環境について説明します．

3.1. 前提とする知識

Nirvana プラグインの開発者は Java のアプリケーション開発に必要な一般的な知識と，Nirvana に関するユーザとしての一般的な知識を有していることを前提とします．

3.2. 開発環境とビルド

基本的な開発環境は 2.2 節の動作環境に示した OS と JDK(Java Development Kit)に準じます．それ以外については特に定めていなません．コンソール(コマンドライン)と Apache Ant を用いる，統合開発環境の Eclipse を用いるなど，どのような Java 開発環境を用いても良いです．なお，個々の開発環境の使用方法については本文書では言及しません．

Nirvana プラグインは以下の手順でビルドできます．ビルドに必要な情報は以下の手順で示した情報だけです．なお，JAR ファイル内部の構成については 4.2.2 項で扱います．

(1) 開発環境を用意します．
(2) Nirvana 本体の JAR ファイル(nirvana.jar)を用意します．
(3) nirvana.jar をクラスパスに指定して，プラグインのソースコードをコンパイルします．
(4) プラグインの JAR ファイルを作成します．

4. Nirvanaプラグインの作成

本章では，プラグイン API の概要，リソースファイル，各機能の具体的な実装方法について説明します．プラグインの作成に必要な要点の説明までとし，各クラスやインターフェースの詳細までは説明しません．各クラスやインターフェースの詳細については，参考資料(1)の Nirvana プラグイン API リファレンスを参照して下さい．

4.1. プラグインAPI

Nirvana プラグインの作成には，プラグイン作成向けに公開されている Nirvana のプラグイン API を使用します．プラグイン API は Nirvana 本体の JAR ファイル(nirvana.jar)に含まれています．

Nirvana プラグイン API は，jp.co.nil.nirvana.pluginapi パッケージ以下に含まれる，クラス，インターフェース，また，それらのスーパークラス，そのスーパーインターフェースです．スーパークラスとスーパーインターフェースは jp.co.nil.nirvana.pluginapi 以外のパッケージに属する場合もありますがそれは使用して良いものです．これらをベースにクラスを作成してプラグインを作成します．

以下に，Nirvana プラグイン API のパッケージ構成の概要を図示します．jp.co.nil.nirvana.pluginapi 以外のパッケージについては省略しています．

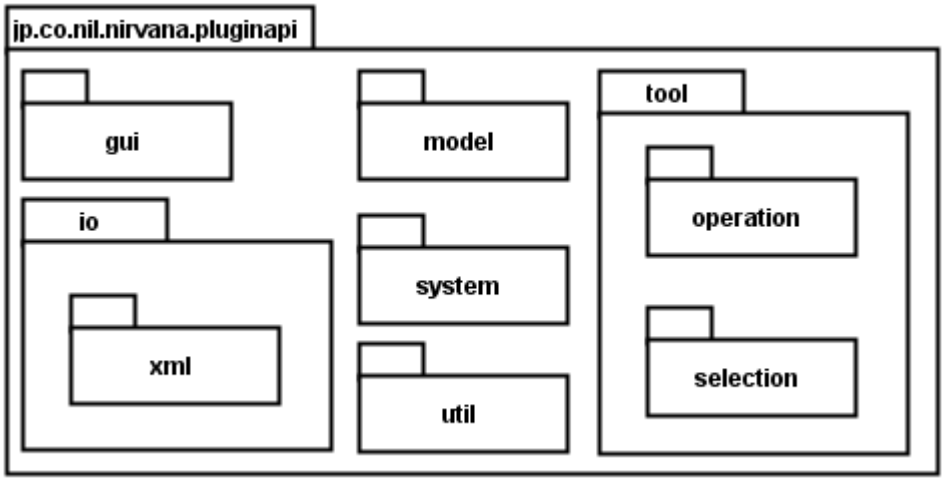

図 4.1. Nirvana プラグイン API のパッケージ構成概要

各パッケージについて概要を説明します．

(1) jp.co.nil.nirvana.pluginapi.gui
Nirvana のメイン画面に関する I/F としてのクラス．環境設定に設定項目を追加する際や，プロパティービューの拡張の際に使用する GUI に関するクラスのスーパークラスやインターフェースを配置しているパッケージです．

(2) jp.co.nil.nirvana.pluginapi.io.xml
図の保存や読込，インポート，エクスポートなど XML を用いた IO に関するクラスのスーパークラスやインターフェースを配置しているパッケージです．なお，Nirvana の IO では XML 形式を使用しています．

(3) jp.co.nil.nirvana.pluginapi.model
図や，図に配置する描画要素などのモデル情報のクラスのスーパークラスやインターフェースを配置しているパッケージです．

(4) jp.co.nil.nirvana.pluginapi.system
Nirvana 本体側の操作を行うための I/F をまとめたクラスを配置しているパッケージです．ここにあるクラスはサブクラスを作らず，そのまま使用します．

(5) jp.co.nil.nirvana.pluginapi.tool
図の編集操作に関するスーパークラスやインターフェースを配置しているパッケージです．

(6) jp.co.nil.nirvana.pluginapi.tool.operation
図上の編集操作に関するスーパークラスやインターフェースを配置しているパッケージです．

(7) jp.co.nil.nirvana.pluginapi.tool.selection
図上の編集操作で，特に描画要素を選択している際に関するスーパークラスやインターフェースを配置しているパッケージです．

(8) jp.co.nil.nirvana.pluginapi.util
汎用的なユーティリティクラス，リソースクラスのスーパークラス，クリップボードに関するクラスを配置しているパッケージ．

(9) その他の jp.co.nil.nirvana.以下のパッケージ
jp.co.nil.nirvana.pluginapi 以下に含まれるクラスやインターフェースの，スーパークラスやスーパーインターフェースのみ Nirvana プラグイン API として公開しています．具体的にどのクラスやインターフェースを公開しているかは添付資料(1)を参照して下さい．

なお，プラグインの作成に必須となるクラスやインターフェースの使用方法については次節以降で機能別に説明します．

4.2. プラグインの作成手順

プラグインの作成手順の概要は以下になります.
(1) リソースファイルに表記法の定義を記述, クラスを実装します.
(2) リソースファイルに図や図の描画要素を定義を記述, クラスを実装します.
(3) リソースファイルに各機能のメニュー(アクション)を定義を記述, クラスの実装します.

次節以降で各手順について説明します.

4.3. プラグインのファイル構成

プラグインのパッケージ名, JAR ファイルについて説明します.

4.3.1. プラグインのパッケージ名

プラグインパッケージ名は, Java のパッケージ名の命名規則に従い,「プラグイン開発者を識別するドメイン表記部」+「nirvanaplugin」+「表記法名」とし, これを 1 つのプラグインの単位とします. 以降では, これを「プラグインパッケージ名」と呼びます. なお, プラグイン開発者を識別するドメイン表記部に「-」が含まれる場合は「_」に置換します.

例) 株式会社ニルソフトウェア(ドメインが nil.co.jp)が, NewMethod という表記法名のプラグインを作成する場合のプラグインパッケージ名
jp.co.nil.nirvanaplugin.NewMethod

4.3.2. プラグインのJARファイルと名前固定なクラス

プラグインファイルは 1 つの JAR ファイルとします. プラグインの JAR ファイルのファイル名は「プラグインパッケージ名」+「_バージョン番号.jar」とします.

例） 株式会社ニルソフトウェア(ドメイン名は nil.co.jp)が NewMethod という表記法名のプラグインのバージョン 1.0 を作成する場合のプラグインの JAR ファイル名
jp.co.nil.nirvanaplugin.NewMethod_1.0.jar

Nirvana がプラグインの機能を使用するにあたり, いくつかの名称を規定しているパッケージと, 名称と所属するパッケージを規定しているクラスがあります. これらについてパッケージツリーの構成を図 4.3.2.に示します.

```
/
 |- プラグインパッケージ名の「.」をファイルセパレータとしたフォルダ階層
      |- resource.properties
      |- resource_ja.properties
      |- Resource.class
      |
      |- images/
      |
      |- gui
      |    |- PropertyViewPane.class
      |    |- GridSetupPanel.class
      |
      |- io
      |    |- NirvanaWriterFactory.class
      |    |- NirvanaHandlerFactory.class
      |
      |- model
      |    |- MethodologyFolder.class
      |
      |- tool
           |- ActionTool.class
           |- DiagramTool.class
           |- OperationFactory.class
```

図 4.3.2. パッケージツリーの構成

各フォルダとクラスファイルについて説明します.

(1) プラグインパッケージのフォルダには, リソースファイル(resource.properties)を置きます. resource.properties ファイルにそのプラグインに関するリソース情報を記述します. 日本語に関するリソース情報は resource_ja.properties ファイルに記述します. 他の言語向けは, resoruce_+「対応する ISO639 の 2 文字の言語コード.proprties」というファイル名で置きます. Nirvana 本体では, resource.properties を読込後, 起動時の環境の言語に対応するリソースファイルを読込み, 同じキーの値を上書きします. リソースファイルの詳細については 4.4.1 項を参照して下さい. 表記法名フォルダには Resouce.class を置きます. このファイルは必須です. これは前述のリソースへのアクセスを提供するクラスです. 詳細は 4.4.2 項を参照して下さい.

(2) images フォルダ以下に, 画像ファイルを置きます.
ツールバーや, Nirvana 本体右上のモデルツリービューに表示するアイコンの画像ファイルを置きます.

(3) guiフォルダには GUI に関するクラスを置きます. PropertyViewPane.class は, Nirvana 本体の右下に表示されるプロパティビューを構成するクラスです. このクラスは必須です. GridSetupPanel.class は, 図のシートレイアウトのグリッドサイズ設定パネルを構成するクラスです. これは必須ではありません. 詳細は 4.10 節を参照して下さい.

(4) io フォルダには, 入出力に関するファイルを置きます. NirvanaWriterFactory.class は図の保存に関するクラスです. NirvanaHandlerFactory.class は図の読込に関するクラスです. これらの 2 クラスは必須です. 詳細は 4.7 節, 4.8 節を参照して下さい.

(5) model フォルダには, 表記法のフォルダを意味する MethodologyFolder や, 図や描画要素などモデルに関するクラス(jp.co.nil.nirvana.pluginapi.model パッケージの PVersionedDiagram, PSymbolElement, PRelationshipElement クラスを継承したクラス)などを置きます. 詳細は 4.6.2.1 目を参照して下さい.

(6) tool フォルダには, 図の表示や図の操作に関するクラスを置きます. ActionTool.class はメニューやキー操作やマウス操作に紐付けられたアクションオブジェクトを返すクラスです. 詳細は 4.4 節を参照して下さい. DiagramTool.class は, 図に対するビューを返すクラスです. OperationFactory.class は, 選択, 貼付け, 移動などの操作オブジェクトを返すクラスです. これらの 3 クラスは必須です. 詳細は 4.6.2.6 目から 4.6.2.8 目を参照して下さい. さらに, tool フォルダに operation, selection フォルダを作成し, 操作に関するクラス(jp.co.nil.nirvana.pluginapi.tool.operation のクラスを継承したクラス)は operation フォルダに, 選択に関するクラス(jp.co.nil.nirvana.pluginapi.tool.selection のクラスを継承したクラス)は selection フォルダに置き, カテゴリ分けすることを推奨します.

4.4. リソース

プラグインの固定的な定義情報，つまり，メニュー定義，メッセージ，使用する画像のファイル名などはリソースとして管理します．それらはリソースファイルにキーと値のリストとして記述します．Nirvana 実行時にそのリソースファイルを読込み，プラグインで作成する Resource クラスを用いてそれらの値を取得します．

4.4.1. リソースファイル

リソースファイルとは，プラグインの情報を Java の Property ファイルの XML ファイルの形式で記述したファイルです．キーと値の組で定義し，リソースファイルに記述した値は，キー文字列を用いて，プラグインパッケージ名.Resource クラスから取得することができます．キー文字列は，リソースファイル内で一意である必要があります．各 entry タグの記述順番は順不同です．

なお，resource.properties にはデフォルトの値を記述します．日本語に関するリソース情報は resource_ja.properties ファイルに記述します．他の言語向けは，resoruce_+「対応する ISO639 の 2 文字の言語コード.proprties」というファイル名で置きます．Nirvana 本体では，resource.properties を読込後，起動時の環境の言語に対応するリソースファイルを読込み，同じキーの値を上書きします．

例）キーと値の定義

resource.properties

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="key1">key1 value</entry>
  <entry key="key2">key2 value</entry>
</properties>
```

resource_ja.properties

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="key2">キー2 の値の日本語表記</entry>
</properties>
```

4.4.1.1. 予約語

キーの先頭に「p.」が付いているキーはNirvanaプラグインの予約されたキーです. プラグイン作成者は「p.」で始まるキーを独自に定義してはいけません.

予約済みのキーと値の説明を一覧表に示します.

表 4.4.1.1.-1.に示したキーは将来的にNirvana本体側でプラグインの情報を表示する際に使用するキーです.

表 4.4.1.1.-1. プラグインの基本情報に関する予約済みのキー

キー	値の説明	必須, 選択
p.name	プラグインの名前	必須
p.version	プラグインのバージョン番号	必須
p.copyright	プラグインの権利者名	必須
p.comment	プラグインのコメント	必須
p.link	プラグインのリンクURL. p.linkはその表示においてURLのリンクとしてブラウザを起動する機能を提供する予定です.	必須
p.kind	プラグインの拡張する機能の種別を示す値を記す. プラグインの将来的な拡張性の為に存在する. 表記法の拡張は「method」, インポート機能の拡張は「import」, エクスポート機能の拡張は「export」を「,」区切りで記述する.	必須

表 4.4.1.1.-2.に示したキーは, メニューとツールバーの拡張に関するキーです. それぞれの詳細は「詳細」列の節を参照して下さい.

表 4.4.1.1.-2. メニューとツールバーに関する予約済みのキー

キー	値の説明	必須, 選択	詳細
p.methodology.folder.name	Nirvana プロジェクト直下の表記法フォルダの名前	必須	4.6 節
p.menu.file.new.diagram.add	図新規作成メニューに追加するメニュー項目のキー	必須	4.7 節
p.menu.file.import.xmi.add	XMI インポートメニューに追加するメニュー項目のキー	選択	4.12 節
p.menu.file.export.add	エクスポートメニューに追加するメニュー項目のキー	選択	4.13 節
p.menu.edit.add	編集メニューに追加するメニュー項目のキー	選択	4.10 節
p.menu.diagram.add	図メニューに追加するメニュー項目のキー	選択	4.10 節
p.menu.symbol.items	シンボルメニューに追加するメニュー項目のキー	必須	4.7.1 項
p.menu.relationship.items	関係メニューに追加するメニュー項目のキー	必須	4.7.1 項
p.toolbar.items	ツールバーに追加するアイコン項目のキー	必須	4.7.1 項

表 4.4.1.1.-3.に示したキーは，管理情報の拡張に関するキーです．それぞれの詳細は「詳細」列の節を参照して下さい．

表 4.4.1.1.-3. 管理情報に関する予約済みのキー

キー	値の説明	必須，選択	詳細
p.project.property.projecttype.add	追加するプロジェクトタイプの選択肢のキー	選択	4.15 節
p.project.property.projectref.add	参照プロジェクト設定項目のキー	選択	4.16 節
p.projectProperty.add	プロジェクトのプロパティダイアログに追加するクラス名のリスト．jp.co.nil.nirvana.pluginapi.gui.CustomProjectPropertyPanel を継承したクラスを指定します．	選択	4.17 節
p.systemsetup.add	環境設定ダイアログに追加するパネルのキー	選択	4.14 節

4.4.1.2. ユーザ定義

リソースファイルでは，4.4.1.1.目で説明した予約語以外にも，メニューの定義や，描画要素の定義を行います．

(1) メニュー定義

リソースファイルで，メニュー項目の定義を行う際に共通する事柄を説明します．

メニュー項目の値に使用した文字列は，そのメニュー項目のキーとなります．
そのキーに，「.label」，「.menuLabel」，「.shortDescription」，「.menuShortcut」を追加したものをキーとし，値を指定することで，そのメニュー項目の表示文字列や詳細説明，ショートカットキーを指定できます．そして，resource_ja.properties では「.label」，「.menuLabel」，「.shortDescription」の値を日本語にしたものを記述すれば，メニュー項目や説明を日本語にすることが出来ます．なお，そのメニュー項目を押下した際にダイアログを表示しまする場合は，メニュー項目名の最後は「...」という表記にします．また，メニューの区切りを追加する場合は，メニュー項目のキー文字列の間に「 - 」を記述します．なお，「.menuShortcut」によるショートカットキーの指定は必須ではありません．ショートカットキーについての詳細は 4.4.1.3 目をご覧下さい．

例) メニュー項目を追加する場合

英語環境で実行時のメニュー階層
File → Export → Added menu B

日本語環境で実行時のメニュー階層
ファイル → エクスポート → 追加するメニュー項目 B

Windows は「Ctrl + B」キー，Mac OS X は「command + B」キーが，ショートカットキーになります．

resource.properties

```
<entry key="p.menu.file.export.add">menuB</entry>
<entry key="menuB.menuShortcut">B</entry>
<entry key="menuB.label">Added menu B</entry>
<entry key="menuB.shortDescription">Description of Added menu B</entry>
```

resource_ja.properties

```
<entry key="menuB.label">追加するメニュー項目 B</entry>
<entry key="menuB.shortDescription">メニュー項目 B の説明</entry>
```

メニューを階層的に定義する場合は次のように定義する.「メニュー階層キー名」は, メニュー階層やキーとなる一意の文字列です.

(1) キー「メニュー階層キー名.menuItemType」の値に「menu」を指定
(2) キー「メニュー階層キー名.menuLabel」にメニュー階層の表示文字列を指定
(3) キー「メニュー階層キー名.items」に格納するメニュー項目のキー文字列を空白区切りで指定

例) メニュー階層とメニュー項目を追加する場合

英語環境で実行時のメニュー階層

```
File → Export → Added Menu B → Added menu C
                                ---------------- ←区切り線
                                Added menu D
```

日本語環境で実行時のメニュー階層

```
ファイル → エクスポート → 追加するメニュー階層 B → 追加するメニュー項目 C
                                                  ------------- ←区切り線
                                                  追加するメニュー項目 D
```

resource.properties

```
<entry key="p.menu.file.export.add">menuB</entry>
<entry key="menuB.menuItemType">menu</entry>
<entry key="menuB.menuLabel">Added Menu B</entry>
<entry key="menuB.items">menuC - menuD</entry>
<entry key="menuC.label">Added menu C</entry>
<entry key="menuC.shortDescription">Description of Added menu C</entry>
<entry key="menuD.label">Added menu D</entry>
<entry key="menuD.shortDescription">Description of Added menu D</entry>
```

resource_ja.properties

```
<entry key="menuB.menuLabel">追加するメニュー階層 B </entry>
<entry key="menuC.label">追加するメニュー項目 C</entry>
<entry key="menuC.shortDescription">メニュー項目 C の説明</entry>
<entry key="menuD.label">追加するメニュー項目 D</entry>
<entry key="menuD.shortDescription">メニュー項目 D の説明</entry>
```

(2) 描画要素の定義

描画要素の線色, 背景色, 線の太さ, フォント, 描画要素を選択した際のポップアップメニューの定義などを定義します. 詳細は 4.7.1 項を参照して下さい.

(3) その他

先頭に「p.」が付かないキーであれば, プラグイン作成者が自由にキーと値を記述してかまいません. ただし, リソースファイル内でキーは一意に識別できる必要があります.

4.4.1.3. メニューのショートカットキーについて

リソースに記述したメニュー項目にはショートカットキーを指定できます．なお，ショートカットキーの指定は必須ではありません．

4.4.1.2 目(1)で説明しました，メニュー項目のキーに「.menuShortcut」を付けたものをキーとし，この値としてキーの組み合わせを記述することでショートカットキーを指定します．また，ショートカットキーの設定が Nirvana 本体や他のプラグインと重複した場合は後から読込まれた設定が有効になるため，必ずしも指定が有効になるとは限りません．

ショートカットキーの値の記述方法を説明します．なお，OSに依存するメニュー修飾キーとは，WindowsではCtrlキー，MacではCommandキーを指します．詳細はJavaのAPIをご覧下さい．[1]

(1) 「OS に依存するメニュー修飾キー＋英数キー」を指定する場合

英数文字を 1 文字記述します．

例) Windows では「Ctrl + B」，Mac では「Command + B」で，このメニュー項目が実行されます
<entry key="p.menu.file.export.add">menuB</entry>
<entry key="menuB.menuShortcut">B</entry>

(2) 英数以外のキーを指定する場合

英数以外のキーは，Java のクラスである java.awt.event. KeyEvent の VK_{文字列}というフィールド名の{文字列}の部分を指定します．例えば，ENTER キーの場合は，java.awt.event. KeyEvent#VK_ENTER の VK_の後の部分，つまり，「ENTER」を指定します．

例)「Enter」を指定する場合
<entry key="p.menu.file.export.add">menuB</entry>
<entry key="menuB.menuShortcut">Enter</entry>

(3) 複数のキーを指定する場合

複数のキーを指定する場合はキーとキーの間に「 + 」を指定します．なお．英数キーの複数指定は最後に指定した英数キーが有効になり，それ以外は無視されます．

例)「OS に依存するメニュー修飾キー + Shift + A」を指定する場合
<entry key="p.menu.file.export.add">menuB</entry>
<entry key="menuB.menuShortcut">MenuSKey + SHIFT + A</entry>

(4) OS に依存するメニュー修飾キーを明示的に指定する場合

OS に依存するメニュー修飾キーを明示的に指定する場合は，「MenuSKey」を指定します．

例)「OS に依存するメニュー修飾キー + Shift + A」を指定する場合
Windows では「Ctrl + Shift + A」，Mac では「Command + Shift + A」になります．
<entry key="p.menu.file.export.add">menuB</entry>
<entry key="menuB.menuShortcut">MenuSKey + SHIFT + A</entry>

[1] Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()

なお, OS の習慣的なキー割り当てに依存するキー設定は, Nirvana 側で自動的に変換します.

表 4.4.1.3.-1. キー割り当ての自動変換

キー割り当ての値	Windows, Linux キーの自動変換	Mac OS X キーの自動変換
MenuSKey	Ctrl	Command
F1	F1	Command + ?
F2	F2	Enter
F5	F5	Command + R

例) Windows では「F5」で, Mac では「Command + R」として扱われます

```
<entry key="p.menu.file.export.add">menuB</entry>
<entry key="menuB.menuShortcut">F5</entry>
```

4.4.2. リソースクラス

リソースファイルに記述した値をキー文字列を用いて取得するためのクラスを作成します.
jp.co.nil.nirvana.pluginapi.util.PluginResource を継承したクラスを, プラグインパッケージ名. Resource として作成します. このクラスは必須です. リソースファイルは Nirvana 起動時に Nirvana 本体側で自動的に読込みます.

4.5. アクション

Nirvana ではユーザによる操作はアクションという単位で扱います. メニューの選択やショートカットキー操作から開始する各処理は, そのメニュー項目のキーに対応するアクションオブジェクトの処理が呼び出され, ここが処理の基点となります.

アクションの追加にはリソースの定義と, クラスの実装が必要です.

4.5.1. リソースの定義

4.4.1.2 目(1)のメニュー項目のキーの文字列を, 各アクション固有のキー文字列として使用します.

4.5.2. クラスの実装

アクションクラスは jp.co.nil.nirvana.pluginapi.tool.PNAction を継承したクラスを作成します. アクションオブジェクトには個々のアクション固有の文字列を保持します. これはリソースファイルのメニュー項目のキー文字列を使用します. このキー文字列はアクションクラスのコンストラクタの引数に指定します. メニューの選択やショートカットキーの操作が行われた際には, アクションクラスの actionPerformed()メソッドが呼び出されます.

さらに, プラグインの全てのアクションオブジェクトを返すクラスを作成します.
jp.co.nil.nirvana.pluginapi.tool.ActionTool を継承したクラスを, プラグインパッケージ名.tool.ActionTool として作成します. このクラスの getActions()メソッドで, プラグインの全てのアクションを取得出来るように実装します.

なお, アクションオブジェクトの数は多くなる傾向があります. そのため, ジャンルごとに別のクラスで生成し, 最終的には, プラグインパッケージ名.tool.ActionTool から取得出来るようにするという実装にすると見通しが良くなります. 例としては, ファイルメニューで使用するアクションは FileActionTool クラスから生成します. 次に, 描画要素を図に追加するアクションは InsertActionTool クラスから生成し, 描画要素を編集する際のアクションは SymbolActionTool から生成します. そして, プラグインパッケージ名.tool.ActionTool の getActions()メソッドから, これら 3 つクラスの getActions()メソッドを呼び出すといった実装が考えられます.

4.6. 表記法の追加

表記法を追加するには，リソースの定義とクラスの実装が必要です．リソースの定義と，クラスの実装を順に説明します．

4.6.1. リソースの定義

リソースの定義として，リソースファイルには「p.methodology.folder.name」キーの値として表記法名を記述します．この名前が，Nirvana プロジェクト内においてその表記法のトップフォルダとして使用されます．

例) MethodologyName という表記法を追加する場合

resource.properties

<entry key="p.methodology.folder.name">MethodologyName</entry>

なお，リソースファイルのこの値については，resource.properties と resource_ja.properties で異なる文字列を指定することは次の理由から推奨しません．異なる文字列を使用した場合，Nirvana 実行時の言語環境により，Nirvana プロジェクト内の表記法のフォルダ名が異なることになります．例としては日本語環境で実行した場合のデータと，英語環境で実行した場合のデータでは，表記法フォルダ名が異なり，その表記法のデータに互換性がないことになってしまいます．

4.6.2. クラスの実装

クラスの実装として，jp.co.nil.nirvana.pluginapi.model.PMethodologyFolder を継承したクラスを，プラグインパッケージ名.model. MethodologyFolder として作成します．これは表記法フォルダを意味するクラスです．

このクラスの protected void setSupportDiagrams()を実装します．具体的には，
jp.co.nil.nirvana.model. MethodologyFolder#setSupportDiagram(String ext, String classpath)
を呼び出すように実装します．これはNirvanaプロジェクトファイルを展開した状態における図に相当するファイルの拡張子と図のクラス名を指定するものです．classpath には 4.7 節で作成する図のクラス名(パッケージ付き)を指定します．拡張子は「.properties」や一般的に使用されている拡張子以外で，かつ，その表記法で使用する拡張子の中でユニークな文字列とする．3 文字から 4 文字を推奨する．

4.7. 図の追加

図を追加するにはリソースの定義とクラスの実装が必要です. リソースで図と描画要素, メニューを定義し, クラスを実装します. リソースの定義と, クラスの実装を順に説明します.

4.7.1. リソースの定義

リソースファイルには, シンボルメニューや関係メニューやツールバーに追加する描画要素のキー一覧を定義します. また, 各描画要素の表示名, 線色, 背景色, 線の太さ, フォント, アイコンのイメージファイル名, 描画要素選択時のポップアップメニューのメニュー階層名と表示しまするメニュー項目も定義します. ここで指定する画像は Nirvana 右上のモデルツリービューや上部のツールバーに表示する際のアイコン画像です.

(1) 4.7.2.4 目で説明する図の新規作成について, 図の新規作成メニュー項目の定義を記述します. 「p.menu.file.new.diagram.add」キーの値として図を示しますキー文字列を記述する. その図を示しすキー文字列に対して「.label」を追加した文字列をキーとしてメニューの項目名を記述します. 同様に「.shortDescription」を追加した文字列をキーとしてメニューの項目の説明を記述します.

例) リソースファイルの記述

resource.properties

```
<entry key="p.menu.file.new.diagram.add">diagramA</entry>
<entry key="diagramA.label">DiagramA</entry>
<entry key="diagramA.shortDescription">Create New DiagramA</entry>
```

resource_ja.prperties

```
<entry key="diagramA.label">図 A</entry>
<entry key="diagramA.shortDescription">図 A を作成する</entry>
```

(2) 4.7.2.1 目で説明する描画要素の定義と, 4.7.2.5 目で説明する描画要素の新規作成, 4.7.2.6 目で説明する描画要素の選択の定義について説明します.

4.7.2.1 目で説明する描画要素の定義として, 描画要素の描画要素のキー名を接頭語とする以下を記述します.

(a) 表示名は「描画要素のキー名.label」をキーとしてを記述します.
(b) 背景色は「描画要素のキー名.backgroundColor」をキーとして記述します.
(c) 線色は「描画要素のキー名.foregroundColor」をキーとして記述します.
(d) 線の太さは「描画要素のキー名.lineThickness」をキーとして記述します.
(e) フォントは「描画要素のキー名.font」をキーとして記述する. フォント指定の文字列は Java のフォント名指定方法に準じます.
(f) アイコンのイメージファイル名は「描画要素のキー名.image」をキーとして記述します.

4.7.2.5 目で説明する描画要素の新規作成操作の定義として, 描画要素の描画要素のキー名を接頭語とする以下を記述します.

(a) 描画要素を図に追加する際のアクションの説明は, 「描画要素のキー名.shortDescription」をキーとして記述します.
(b) メニューの定義を記述します. シンボルのメニューは, 「p.menu.symbol.items」をキーにし, 関係のメニューは, 「p.menu.relationship.items」をキーにし, ツールバーに追加するシンボルや関係線は共に「p.toolbar.items」をキーにし, 値に, 描画要素のキー名を, 空白区切りで記述します.

4.7.2.6 目で説明する描画要素の選択について，描画要素の描画要素のキー名を接頭語とする以下を記述します．

(a) 描画要素選択時のポップアップメニューのメニュー階層名は「描画要素のキー.selection.menuLabel」をキーとして記述します．

(b) 描画要素選択時のポップアップメニューの表示しまするメニュー項目は，キー「描画要素のキー.selection.items」をキーにし，値には表示するメニュー項目のキー名を記述します．この値には，デフォルトで用意されている「changeElementBackgroundColor」と「changeElementLineColor」を使用できます．「changeElementBackgroundColor」はその描画要素の背景色を変更するアクション，「changeElementLineColor」はその描画要素の線の色を変更するアクションを意味します．

例) プラグインパッケージ名が jp.co.nil.nirvanaplugin.NewMethod，シンボルメニューにシンボル A とシンボル B を追加，関係メニューに関係線 A と関係線 B を追加．ツールバーにシンボル A とシンボル B と関係線 A と関係線 B を追加する場合の例．なお，シンボル B，関係線 A，関係線 B については省略しています．また，symbolA.selection.items の値の箇所は，本来は改行しませんが文書において見易くするため改行しています．

```
resource.properties
    <entry key="p.menu.symbol.items">symbolA symbolB</entry>
    <entry key="p.menu.relationship.items">relationshipA relationshipB</entry>

    <entry key="p.toolbar.items">symbolA symbolB - relationshipA relationshipB</entry>

    <entry key="symbolA.label">Symbol A</entry>
    <entry key="symbolA.backgroundColor">#ffffff</entry>
    <entry key="symbolA.foregroundColor">#000000</entry>
    <entry key="symbolA.lineThickness">1</entry>
    <entry key="symbolA.font">SansSelf-plain-12</entry>
    <entry ey="symbolA.image">/jp/co/nil/nirvanaplugin/NewMethod/images/sA.png</entry>
    <entry key="symbolA.shortDescription">Add Symbol A</entry>
    <entry key="symbolA.selection.menuLabel">Symbol A</entry>
    <entry key="symbolA.selection.items">
                    changeElementBackgroundColor changeElementLineColor</entry>

resource_ja.properties
    <entry key="symbolA.label">シンボル A</entry>
    <entry key="symbolA.shortDescription">シンボル A を追加</entry>
    <entry key="symbolA.selection.menuLabel">シンボル A</entry>
```

4.7.2. クラスの実装

MVC に従い，図の定義にはまず以下の 3 つのクラスが必要です．

(1) モデルとして，jp.co.nil.nirvana.pluginapi.model.PVersionedDiagram を継承したクラスをプラグインパッケージ名.model に作成します．これは図の情報（図に属する描画要素なども含む）を保持するクラスです．メソッド getDiagramType()，getFileExtension()，getImageIcon()の実装は必須です．他にも適宜必要なメソッドをオーバーライドします．

(2) ビューとして，jp.co.nil.nirvana.pluginapi.tool.PDiagramView を継承したクラスをプラグインパッケージ名.tool に作成します．これは図の描画に関するクラスです．

(3) コントロールとして，jp.co.nil.nirvana.pluginapi.tool.PDiagramTool を継承したクラスをプラグインパッケージ名.tool.DiagramTool として作成します．これは図のビューを生成して返すクラスです．

さらに，この 3 つのクラスをサポートするクラスを追加します．

モデルのサポートクラスとしては，図の描画要素(シンボル，関係線)クラスがあります．
ビューのサポートクラスとしては，ビューオプションクラス，ビューモニタークラス，プロパティビュークラスがあります．
コントロールのサポートクラスとしては，各操作ごとの操作クラス，各描画要素ごとに選択クラスがあます．
これらの実装方法について以降で順に説明します．

4.7.2.1. 描画要素

描画要素のクラスについて説明します．

シンボルを作成するには jp.co.nil.nirvana.pluginapi.model.PSymbolElement を継承したクラスを作成します．関係線を作成するには jp.co.nil.nirvana.pluginapi.model.RelationshipElement を継承したクラスを作成します．これらのクラスはプラグインパッケージ名.model 以下に作成します．描画要素に共通するスーパークラスは jp.co.nil.nirvana.diagram.ViewElement です．プラグインで作成する描画要素について継承関係を図 4.7.2.1.に示します．

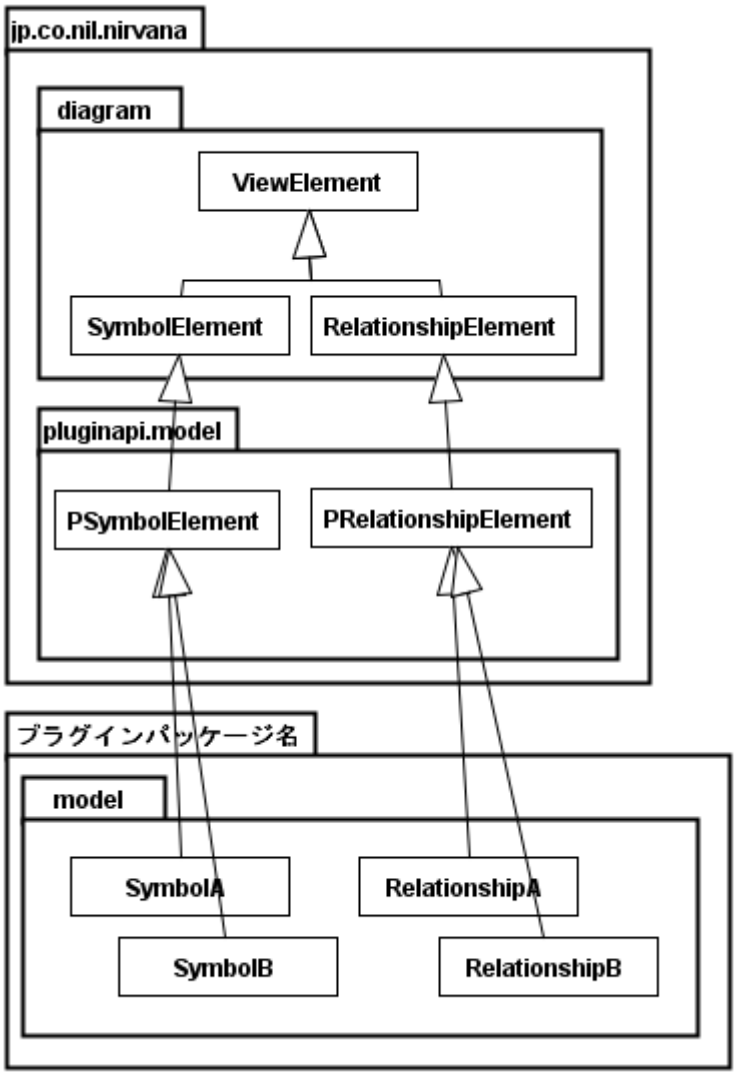

図 4.7.2.1. 描画要素について ViewElement からの継承関係

4.7.2.2. ビューオプション，ビューモニタークラス

4.7.2 項の(2)で作成した PDiagramView を継承したクラスで使用するクラスを実装します．

(1) ビューモニターは，jp.co.nil.nirvana.pluginapi.model.PViewMonitor を継承したクラスを，プラグインパッケージ名.tool に作成します．これは図の描画要素の描画後に処理を行う際に使用する描画モニタークラスです．

(2) ビューオプションクラスは，jp.co.nil.nirvana.pluginapi.model.PViewOption を継承したクラスを，プラグインパッケージ名.tool に作成します．これは図の描画に関するオプションを保持するクラスです．

4.7.2.3. プロパティービュー

Nirvana 本体の画面構成において右下に表示される，選択した描画要素のプロパティービューに使用するクラスを作成します．描画要素を選択された際にその描画要素に合わせたパネルを表示できるようにするものです．

javax.swing.JTabbedPane を継承したクラスを，プラグインパッケージ名.gui .PropertyViewPane として作成します．このクラスは jp.co.nil.nirvana.pluginapi.gui.PPropertyViewPane と TableModelListener を実装します．

4.7.2.4. 図の新規作成メニュー

図の新規作成メニューを追加するには，リソースの定義とアクションクラスの実装が必要です．リソースの定義については 4.7.1 項で説明しました．クラスの実装を説明します．

クラスの実装としては，PNAction を継承したアクションクラスを作成します．そのアクションクラスでは，jp.co.nil.nirvana.pluginapi.system. ModelTreeOperator#createOpenDiagram(String diagramClassName)を用いて実装を行います．なお，このアクションクラスのオブジェクトは，4.4 節で説明したプラグインパッケージ名.tool.ActionTool から取得できるようにします．

4.7.2.5. 描画要素の新規作成

描画要素の新規作成操作を追加するには，リソースの定義とアクションクラスの実装が必要です．リソースの定義については 4.7.1 項で説明しました．クラスの実装を説明します．

クラスの実装としては，まず，4.4 節で説明した通り，各描画要素ごとにアクションクラスを実装します．ここで，アクションクラスのオブジェクトに指定するキー文字列は，4.7.1 項で p.menu.symbol.item，p.menu.relationship.items，p.toolbar.items の値に書いた描画要素のキー文字列を使用します．これらのアクションクラスから次の操作クラスを呼び出して，新規作成操作を開始します．

シンボルの新規作成の操作クラスは，jp.co.nil.nirvana.pluginapi.tool.operation.PInsertOperation を継承したクラスをプラグインパッケージ名.tool.operation 以下に作成します．

関係線の新規作成の操作クラスは，jp.co.nil.nirvana.pluginapi.tool.operation.PCreatePathOperation を継承したクラスをプラグインパッケージ名.tool.operation 以下に作成します．

4.7.2.6. 描画要素の選択

描画要素の新規作成操作については，リソースの定義とクラスの実装が必要です．リソースの定義は 4.7.1 項で説明しました．クラスの実装を説明します．

まず，選択した描画要素に対応する選択状態クラス(以下，Selection クラス)について説明します．
シンボルについては jp.co.nil.nirvana.pluginapi.tool.selection.PSymbolSelection を継承して各シンボルに対応するクラスをプラグインパッケージ名 .tool.selection に作成します．関係線については jp.co.nil.nirvana.pluginapi.tool.selection.PRelationshipSelection を継承して各関係線に対応するクラスをプラグインパッケージ名.tool.selection に作成します．継承関係を図 4.7.2.6.に示します．

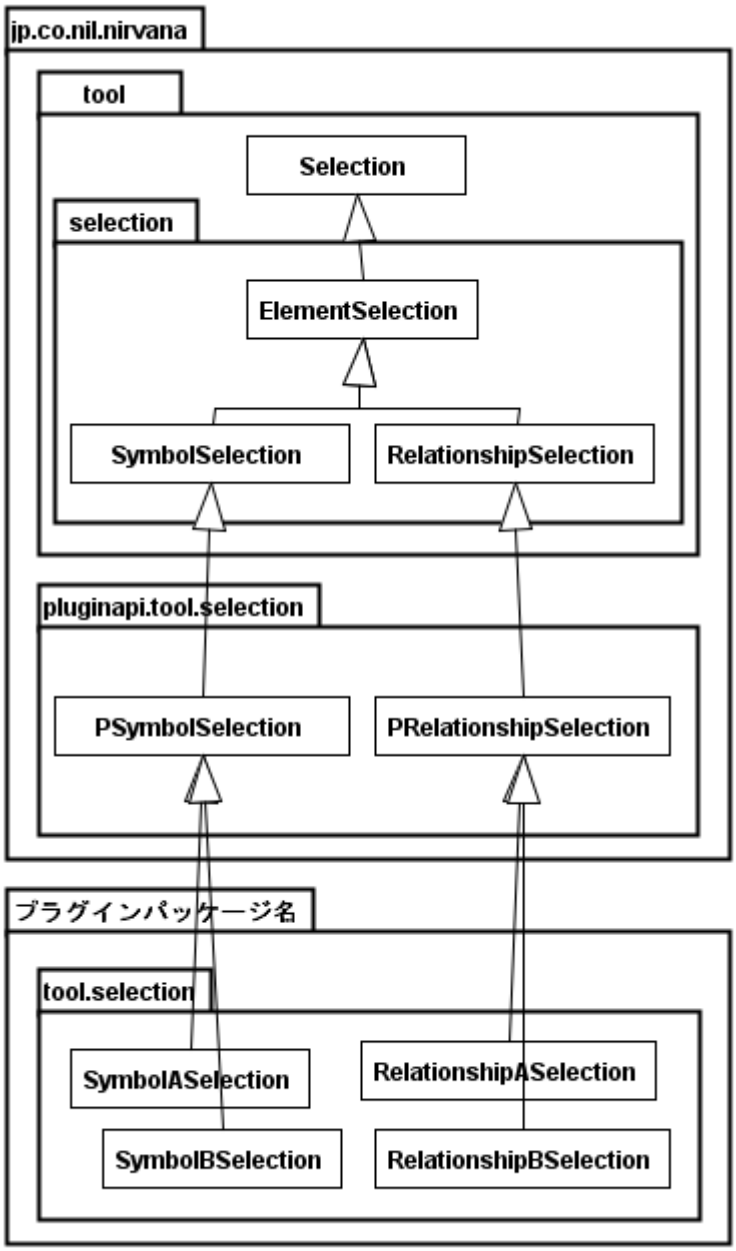

図 4.7.2.6. 描画要素の選択時の操作処理について Selection からの継承関係

そして，今選択している描画要素に対して対応する Selection クラスのサブクラスを返すために，jp.co.nil.nirvana.pluginapi.tool.PSelectionManager を継承したクラスを作成します．このクラスの createSelection()メソッドでは，引数に指定された描画要素を元に対応する Selection クラスのサブクラスのオブジェクトを返すように実装します．なお，このクラスのオブジェクトは，4.7.2 項(2)で説明した PDiagramView を継承したクラスであるビュークラスで管理します．

その後，描画要素を選択する操作のクラス(以下 Operation クラス)を，プラグインパッケージ名.tool.operation に作成します．このクラスは，jp.co.nil.nirvana.pluginapi.tool.operation.PSelectOperation を継承したクラスとして作成します．このクラスには，先の PSelectionManager を継承したクラスを用いて，選択した描画要素に対応する Selection クラスを取得して，その描画要素に対して処理を行うように実装します．

最後に，Nirvana から呼び出される必須クラスとして，プラグインパッケージ名.tool.OperationFactory を実装します．このクラスは，jp.co.nil.nirvana.pluginapi.tool.operation.POperationFactory を継承したクラスとして作成します．このクラスは描画要素の選択，貼付け，移動操作のそれぞれの Operation クラスのオブジェクトを返すファクトリークラスとして実装します．

なお，図上を移動するカーソルの様な機能を実現するには，描画要素を選択していない状態における，jp.co.nil.nirvana.pluginapi.tool.operation.PSelectOperation を継承したクラスとして作成し，適切にプラグインパッケージ名.tool.OperationFactor で返すように実装することで実現できます．

4.7.2.7. 描画要素の貼付け

描画要素の貼付けについては，クラスの実装のみとなります．

jp.co.nil.nirvana.pluginapi. PPasteOperation を継承したクラスをプラグインパッケージ名.tool.operation に作成します．このクラスは initializeImpl()メソッドと keyPressed()メソッドなどをオーバーライドして実装します．

最後に，4.7.2.6 項の最後で説明したプラグインパッケージ名.tool.OperationFactory のメソッドで，PPasteOperation を継承したクラスのオブジェクトを返すように実装します．

4.7.2.8. 描画要素の移動

描画要素の移動については，クラスの実装のみとなります．

jp.co.nil.nirvana.pluginapi. PMoveOperation を継承したクラスを作成します．このクラスは initializeImpl()メソッドなどをオーバーライドして実装します．

最後に，4.7.2.6 項で最後で説明したプラグインパッケージ名.tool.OperationFactory のメソッドで，PPasteOperation を継承したクラスのオブジェクトを返すように実装します．

4.7.2.9. シンボルから，他の図へのリンク，または，URIを用いた外部へのリンク

シンボルは，シンボルから他の図へのリンク，または，URI を用いた外部へのリンクを設定できるように実装できます．

シンボルから他の図へのリンク（以下，図リンクと記す）は，シンボルのポップアップメニューに「リンク」が追加され，そのシンボルから他の図へリンクを追加，変更，削除，リンク先への移動が行えるようになります．

シンボルから URI を用いた外部へのリンク（以下，外部リンクと記す）は，ユーザがシンボルのポップアップメニューに「リンク」とその下位に「外部リンク」が追加され，そのシンボルから，URI やファイルのプロジェクトからの相対パスを指定の追加，変更，削除，その URI やファイルに対応するアプリケーションの起動が行えるようになります．

これらを実現するには，クラスの実装が必要となります．図リンク，外部リンクに分けて説明します．

(a) 図リンク

まず，図リンクを行えるようにするシンボルのクラス（PSymbolElement を継承したクラス）に，jp.co.nil.nirvana.pluginapi.model.PLinkableSymbol インターフェースを実装します．この際，jp.co.nil.nirvana.pluginapi.model.PLinkableSymbolSupport クラスを利用できます．また，PLinkableSymbolSupport#addLinkableDiagramType(DiagramType diagramType)でリンク可能な図種を指定します．この際に，Nirvana 本体や他のプラグインが保持する図種を指定するには，jp.co.nil.nirvana.pluginapi.system.DiagramManager の以下のフィールドやメソッドを用います．

```
DiagramType DIAGRAM_TYPE_USE_CASE_DIAGRAM
DiagramType DIAGRAM_TYPE_CLASS_DIAGRAM
DiagramType DIAGRAM_TYPE_SPEC_MODEL_DIAGRAM
DiagramType DIAGRAM_TYPE_ACTIVITY_DIAGRAM
DiagramType DIAGRAM_TYPE_STATE_DIAGRAM
DiagramType DIAGRAM_TYPE_SEQUENCE_DIAGRAM
DiagramType DIAGRAM_TYPE_COMPONENT_DIAGRAM
public static String getDiagramTypeName(String diagramclass)
public HashMap<String, ArrayList<String>> getAllDiagramKind()
public HashMap<String, ArrayList<String>> getNirvanaDiagramKind()
public HashMap<String, ArrayList<String>> getPluginsDiagramKind()
```

次に，そのシンボルの選択クラス（jp.co.nil.nirvana.pluginapi.tool.selection.PSymbolSelection を継承したクラス）のコンストラクタでは，super(SelectionTool etool, LinkableSelectionTool ltool) と initLinkAction() を呼び出すようにします．また，drawSymbol(Graphics g, PDrawingOption option)では，リンクが存在する場合には，リンクマークを表示するように実装します．その際は，PLinkableSymbolSupport#generateLinkMark()を用います．

さらに，getClickCommand(Point location)をオーバーライドし，シンボル選択時にクリックされた位置を判定し，リンクマークに対応するアクションのキー文字列を返すように実装します．そのキー文字列は，リソースクラス（jp.co.nil.nirvana.pluginapi.util .PluginResource を継承したクラス）の getLinkShowDiagramActionName()を使用します．

以上の実装で，そのシンボルから図リンクを使用できるようになります．

(b) 外部リンク

まず，外部リンクを行えるようにするシンボルのクラス（PSymbolElement を継承したクラス）に，jp.co.nil.nirvana.pluginapi.model.POuterLinkableSymbol インターフェースを実装します．この際，jp.co.nil.nirvana.pluginapi.model.POuterLinkableSymbolSupport クラスを利用できます．

次に，そのシンボルの選択クラス（jp.co.nil.nirvana.pluginapi.tool.selection.PSymbolSelection を継承したクラス）のコンストラクタでは，super(SelectionTool etool, LinkableSelectionTool ltool) と initLinkAction() を呼び出すようにします．また，drawSymbol(Graphics g, PDrawingOption option)では，リンクが存在する場合には，リンクマークを表示するように実装します．その際は，OuterLinkableSymbolSupport#generateLinkMark()を用います．

さらに，getClickCommand(Point location)をオーバーライドし，シンボル選択時にクリックされた位置を判定し，リンクマークに対応するアクションのキー文字列を返すように実装します．そのキー文字列は，リソースクラス（jp.co.nil.nirvana.pluginapi.util .PluginResource を継承したクラス）の MENU_OUTER_LINK_SHOW を使用します．

以上の実装で，そのシンボルから外部リンクを使用できるようになります．

4.8. 図の保存

追加した図の保存にはクラスの実装のみとなります.

4.8.1. クラスの実装

jp.co.nil.nirvana.pluginapi.io.xml .PluginNirvanaWriterFactory を継承した,
プラグインパッケージ名.io. NirvanaWriterFactory というクラスを作成します. これは PNirvanaXMLWriter オブジェクトと, PXMIWriter オブジェクトを返すファクトリークラスです. Nirvana 本体からメソッドを呼び出される形で使用されます. なお, XMI 出力を使用しない場合は, getXMIWriter()メソッドは null を返すように実装します. XMI 出力については4.12 節を参照して下さい.

NirvanaWriterFactory の getNirvanaXMLWriter()メソッドは引数に指定された図のモデルのオブジェクトがプラグインで追加した図であった場合に対応する PXMIWriter オブジェクトを返すように実装します. Nirvana 本体の図のオブジェクトなどが指定された場合は null を返すように実装します.

次に, NirvanaWriterFactory から返すオブジェクトのクラスとして, jp.co.nil.nirvana.pluginapi.io.xml.PNirvanaXMLWriter を継承したクラスを作成し, プラグインパッケージ名.io パッケージに配置します. PNirvanaXMLWriter で abstract として定義されている writeDiagram()メソッドに, 追加した図を XML で出力する実装を行います.
その際, NirvanaXMLWriter の setDiagramWriteMode()メソッドの引数に指定される mode により, 出力対象を限定された場合の考慮を行い実装する必要があります. mode は NirvanaXMLWriter のフィールドに定義されている「DIAGRAM_」から始まる整数値が指定されるます.

4.9. 図の読込

追加した図の読込にはクラスの実装のみになります.

4.9.1. クラスの実装

jp.co.nil.nirvana.pluginapi.io.xml . PluginNirvanaHandlerFactory を継承した,
プラグインパッケージ名.io. NirvanaHandlerFactory というクラスを作成します.

次に, NirvanaHandlerFactory から返すオブジェクトのクラスとして, jp.co.nil.nirvana.pluginapi.io.xml . PNirvanaDiagramHandler を継承したクラスを作成し, プラグインパッケージ名.io パッケージに配置します.
その際, NirvanaDiagramHandler の setDiagramHandleMode ()メソッドの引数に指定される mode により, 読込対象を限定された場合の考慮を行い実装する必要があります. mode は NirvanaDiagramHandler のフィールドにフィールドに定義されている「DIAGRAM_」から始まる整数値が指定されます.

4.10. 追加した図を開いている場合のみ使用する処理の追加

以下のメニュー階層の最後にメニュー項目を追加できます.
(a) メニュー → 編集 →[追加したメニュー項目名]
(b) メニュー → 図 →[追加したメニュー項目名]

まず, リソースファイルにメニューの定義を追加し, 更にそのメニューを実行した際に処理を行うアクションクラスを実装します.

4.10.1.リソースの定義

リソースファイルのメニューの定義について, (a)はキー文字列を「p.menu.edit.add」, (b)はキー文字列を「p.menu.diagram.add」とし, 4.3.1.1 目で説明したようにメニュー項目のキー文字列を指定します.

4.10.2.クラスの実装

対応するアクションクラスは 4.4 節で説明したように適宜作成します. なお, その際にアクションオブジェクトに指定するキー文字列は, 前述のリソースファイルに書いたメニュー項目のキー文字列を使用します.

4.11. シートレイアウトのグリッドサイズ設定機能の変更

Nirvana の図のシートレイアウトのグリッドサイズ設定のパネルを, プラグイン向けのグリッドサイズ設定パネルを置き換えるには, クラスの実装のみとなります.

4.11.1.クラスの実装

まず, 前提として図のグリッドサイズは 4.7.2 項(1)で説明した PVersionedDiagram を継承した図のモデルクラスが保持しています. これを踏まえた上で, jp.co.nil.nirvana.pluginapi.gui. PGridSetupPanel を継承したクラスを, プラグインパッケージ名.gui.GridSetupPanel として作成します.

GridSetupPanel クラスでは, PGridSetupPanel で abstract として定義されているメソッドを全て実装します. まず, コンストラクタで GUI の準備を完了するように実装します. GridSetupPanel の initialize(), reset()メソッドでは, 図のモデルクラスのグリッドサイズ取得メソッドを使用して既存の値を GUI にセットする処理を実装します. また, apply()メソッドでは, 図のモデルクラスのグリッドサイズ格納メソッドを使用して図にグリッドの情報にセットする処理を実装します.

4.12. インポート機能

インポート機能を追加するには，リソースの定義とクラスの実装が必要です．リソースの定義と，クラスの実装を順に説明します．

4.12.1.リソースの定義

リソースの定義として，リソースファイルにインポートメニューに追加するメニュー項目の情報を記述します．

(1) キー「p.menu.file.import.xmi.add」の値に，リソースファイル内で一意となるキー名として，メニュー項目のキー名を指定します．
(2) キー「メニュー項目のキー名」+「.label」の値に，メニュー項目の表示名を指定します．
(3) キー「メニュー項目のキー名」+「.shortDescription」の値に，メニュー項目の表示名を指定します．

その図を示しすキー文字列に対して「.label」を追加した文字列をキーとしてメニューの項目名を記述します．同様に「.shortDescription」を追加した文字列をキーとしてメニューの項目の説明を記述します．

例) リソースファイルの記述

resource.properties

```
<entry key="p.menu.file.export.add">importXmiToDiagramA</entry>
<entry key="importXmiToDiagramA.label"> XMI to Diagram A</entry>
<entry key="importXmiToDiagramA.shortDescription">Import XMI to Diagram A</entry>
```

resource_ja.properties

```
<entry key="importXmiToDiagramA.label">XMI から図 A</entry>
<entry key="importXmiToDiagramA.shortDescription">XMI から図 A をインポート</entry>
```

4.12.2.クラスの実装

クラスの実装としてしては，まず，4.4 節で説明した通り，PNAction を継承したアクションクラスを作成します．このアクションオブジェクトの保持する固有なキー文字列は上記で指定した文字列(上記の例では「importXmiToDiagramA」)を指定します．また，このアクションクラスのオブジェクトは，4.4節で説明したプラグインパッケージ名.tool.ActionTool から取得できるようにします．

なお，この機能についてはアクション以降の処理をプラグイン作成者が全て作成する必要があります．インポート機能で特に XMI を入力とする場合は，XML を解析する実装のサポートとして jp.co.nil.nirvana.pluginapi.io.xml.PXMIHandler クラスを用意しています．このクラスを基底クラスとしてクラスを作成すると実装の手間を一部省けます．

また，プロジェクト内のどの位置(プロジェクト内のフォルダ)にインポートするか選択するといったダイアログを必要とする場合は，プラグイン作成者がそのダイアログ作成する必要があります．ダイアログを作成する際には，PButtonDialog を継承したクラスを作成します．この詳細については 4.11.1 項を参照して下さい．さらに，プロジェクトのどこにインポートした図を配置するかをユーザに問い合わせるために，プロジェクト内のツリー構造を作成するには，プロジェクトの内部構造の取得が必要になります．この場合には，プロジェクトの内部構造を表すクラスである，jp.co.nil.nirvana.pluginapi.model パッケージの PProject クラス，PFolder クラス，PDiagram クラス，PVersionedDiagram クラスについて理解する必要があります．この詳細については 4.11.2 項を参照して下さい．

4.12.2.1. ダイアログの作成

ダイアログを作成する場合は以下のいずれかのクラスを継承して実装します.

(1) jp.co.nil.nirvana.pluginapi.gui.PButtonDialog
　　各種ボタン付きの汎用的なダイアログです.

(2) jp.co.nil.nirvana.pluginapi.gui.PMessageDialog
　　各種メッセージを表示するための専用ダイアログです.

4.12.2.2. Nirvanaプロジェクト内の構造

Nirvana のプロジェクトの内部構造は, jp.co.nil.nirvana.pluginapi.model パッケージの PProject クラス, PFolder クラス, PDiagram クラス, PVersionedDiagram クラス, または, そのスーパークラスを用いて表現されます. クラスの継承関係を図 4.11.2.に示します. それぞれ包含で示された親子関係を辿ることが出来ます. クラスの詳細はリファレンスマニュアルを参照して下さい.

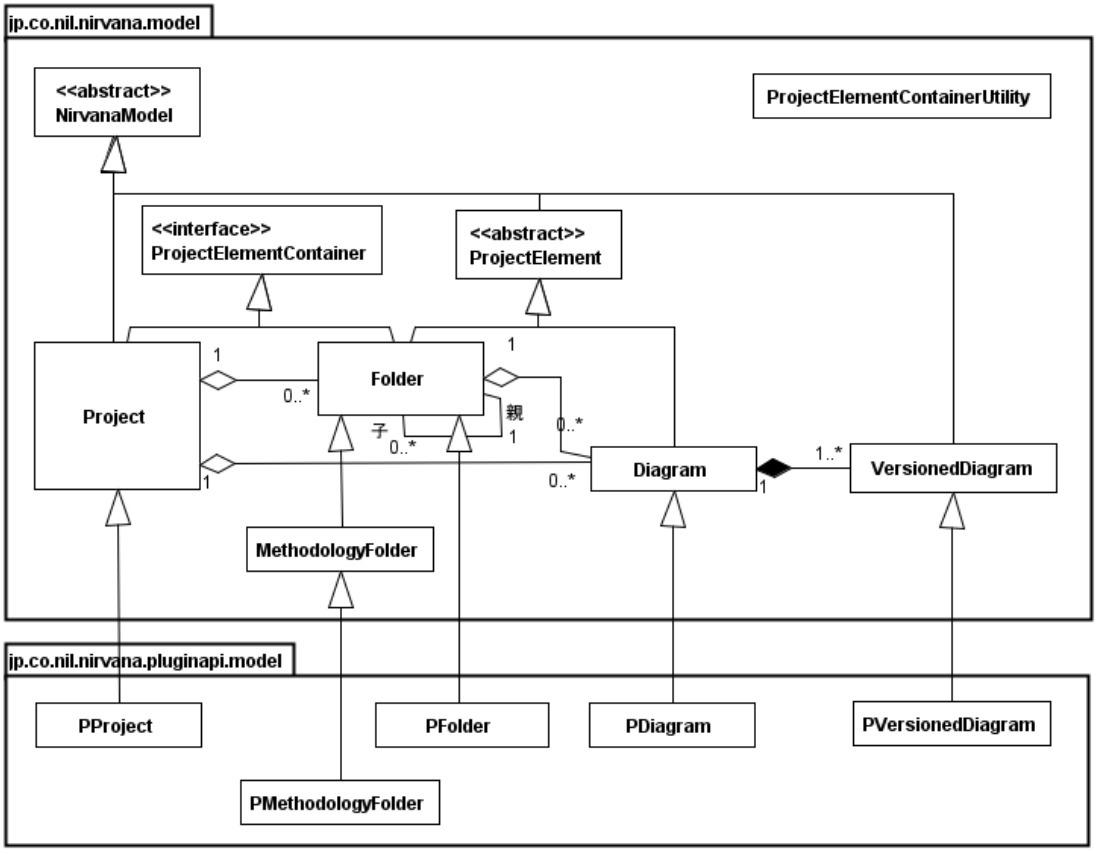

図 4.12.2.2. Nirvana プロジェクト内の構造

4.13. エクスポート機能

エクスポート機能を追加するには，リソースの定義とクラスの実装が必要です．リソースの定義と，クラスの実装を順に説明します．

4.13.1.リソースの定義

リソースの定義として，リソースファイルにエクスポートメニューに追加するメニュー項目の情報を記述します．

(4) キー「p.menu.file.export.add」の値に，リソースファイル内で一意となるキー名として，メニュー項目のキー名を指定します．
(5) キー「メニュー項目のキー名」+「.label」の値に，メニュー項目の表示名を指定します．
(6) キー「メニュー項目のキー名」+「.shortDescription」の値に，メニュー項目の表示名を指定します．

その図を示しすキー文字列に対して「.label」を追加した文字列をキーとしてメニューの項目名を記述します．同様に「.shortDescription」を追加した文字列をキーとしてメニューの項目の説明を記述します．

例) リソースファイルの記述

resource.properties

```
<entry key="p.menu.file.export.add">exportDiagramAToXmi</entry>
<entry key="exportDiagramAToXmi.label">Diagram A to XMI</entry>
<entry key="exportDiagramAToXmi.shortDescription">Export Diagram A to XMI</entry>
```

resource_ja.properties

```
<entry key="exportDiagramAToXmi.label">図 A から XMI </entry>
<entry key="exportDiagramAToXmi.shortDescription">図 A から XMI をエクスポート</entry>
```

4.13.2.クラスの実装

クラスの実装としてしては，まず，4.4 節で説明した通り，PNAction を継承したアクションクラスを作成します．このアクションオブジェクトの保持する固有なキー文字列は上記で指定した文字列(上記の例では「exportDiagramAToXmi」)を指定します．また，このアクションクラスのオブジェクトは，4.4 節で説明したプラグインパッケージ名.tool.ActionTool から取得できるようにします．

なお，エクスポート機能で，特に XMI を出力する場合は，4.7 節で説明した NirvanaWriterFactory クラスが返す，PXMIWriter オブジェクトのクラスを作成し，このクラスから出力するように実装します．つまり，jp.co.nil.nirvana.pluginapi.io.xml .PXMIWriter を継承したクラスを作成します．

4.14. Nirvanaの環境設定にプラグイン向けの設定項目の追加

Nirvana の環境設定にプラグイン向けの設定項目の追加するには，リソースの定義とクラスの実装が必要です．リソースの定義と，クラスの実装を順に説明します．

4.14.1.リソースの定義

キー「p.systemsetup.add」に値としてプラグインパッケージ名.gui パッケージに配置した，設定パネルのクラス名を指定します．このクラスの詳細は 4.13.2 項を参照して下さい．

リソースファイルに以下の記述を行います．
(1) キー「p.systemsetup.add」の値に，リソースファイル内で一意となるキー名として，設定パネルのクラス名を指定します．
(2) キー「設定パネルのクラス名」+「.label」の値に，Nirvana の環境設定ダイアログに表示されるタブ名を指定します．

例）設定パネルのクラス名を「PluginSetupPanel」，Nirvana の環境設定ダイアログに表示されるタブ名を「PluginSetupPanel」，日本語環境では「新表記法設定」と指定する場合

reource.properties
<entry key="p.systemsetup.add">PluginSetupPanel</entry>
<entry key="PluginSetupPanel.label">NewMethod Setup</entry>

reource_ja.properties
<entry key="PluginSetupPanel.label">新表記法設定</entry>

4.14.2.クラスの実装

jp.co.nil.nirvana.pluginapi.gui.CustomSystemSetupPanel を継承したクラスを，
プラグインパッケージ名.gui .NNPluginSetupPanel として作成します．

NNPluginSetupPanel クラスは，コンストラクタ内で全ての GUI の準備を完了するように実装します．その際，プラグインパッケージ名.Resource クラスの getToolSetting()メソッドを使用して，プラグイン向けの設定項目の値を読込して下さい．また，NNPluginSetupPanel クラスは，設定ダイアログが確定して終了した際に Nirvana 本体から applyImpl()メソッドが呼び出されます．applyImpl()メソッドではプラグインパッケージ名.Resource クラスの putToolSetting()メソッドを使用して，プラグイン向けの設定項目の値を保存して下さい．

4.15. プロジェクトのプロパティにプロジェクトタイプの追加

プロジェクトのプロパティ情報でプロジェクトタイプ(プロジェクトの種類)の選択肢を追加できます. これはプロジェクトをいくつかの種類に分けて扱うプラグインを作成する場合などの使用を想定しています.

リソースファイルに定義を記述する事で Nirvana 本体側が解釈し, プロジェクトタイプが追加され, プロジェクトのプロパティで設定可能になります. 以下のメニューで開かれるプロジェクトプロパティダイアログに表示されます. クラスの実装はいりません.
(1) メニュー → プロジェクト → プロパティ

4.15.1.リソースの定義

プロジェクトタイプの予約語は「default」と「tempDiagramProject」です. 「defaut」は Nirvana に標準で用意されている標準プロジェクトを意味します. 「tempDiagramProject」は Nirvana に標準で用意されている一時保存プロジェクトを意味します.

リソースファイルに以下の記述を行います.
(7) キー「p.project.property.projecttype.add」の値に, リソースファイル内で一意となるキー名として, プロジェクトタイプのキー名を指定します. このキーの値にはプロジェクトタイプの予約語以外の任意の文字列を使用できます.
(8) キー「プロジェクトタイプのキー名」+「.label」の値に, プロジェクトのプロパティでプロジェクトタイプの選択肢に表示される項目名を指定します.

例) プロジェクトタイプのキー名を「projectType1」, プロジェクトのプロパティでプロジェクトタイプの選択肢に表示される項目名を「Project Type 1」, 日本語環境では「プロジェクトタイプ 1」と指定する場合.

```
reource.properties
  <entry key="p.project.property.projecttype.add">projectType1</entry>
  <entry key="projectType1.label">Project Type 1</entry>

reource_ja.properties
  <entry key=" projectType1.label">プロジェクトタイプ 1</entry>
```

4.16. プロジェクトのプロパティに参照プロジェクト設定項目の追加

参照プロジェクト設定項目の追加ができます. 参照プロジェクトとは他のプロジェクトを参照する情報の事です. 複数のプロジェクトを組み合わせて何かを行うプラグインを作成する場合などの使用を想定しています.

リソースファイルに定義を記述する事でNirvana本体側のプロパティに, 参照プロジェクト項目が追加され, プロジェクトのプロパティで設定可能になります. クラスの実装はいりません. 以下のメニューで開かれるプロジェクトプロパティダイアログに表示されます.
(1) メニュー → プロジェクト → プロパティ

4.16.1.リソースの定義

リソースファイルに以下の記述を行います.
(1) キー「p.project.property.projectref.add」の値に, リソースファイル内で一意となるキー名として, 参照項目キー名を指定します.
(2) キー「参照項目キー名」+「.label」の値に, プロジェクトのプロパティで表示される項目名を指定します.
(3) キー「参照項目キー名」+「.selectfrom」の値に選択肢として表示するプロジェクトのプロジェクトタイプを指定します. プロジェクトのプロパティでは, ここに指定したプロジェクトタイプのプロジェクトのみ選択肢として表示します. 4.14 節で追加したプロジェクトタイプ名や, 4.14 節で説明した標準プロジェクトタイプの予約語「default」を空白区切りで指定します.

例) 参照項目キー名を「refProject」, プロジェクトのプロパティで参照プロジェクトとして表示される項目名を「Reference Project Setting」, 日本語環境では「参照するプロジェクトの指定」, 選択肢として表示されるプロジェクトのプロジェクトタイプは「projectType1」のみと指定する場合

```
reource.properties
  <entry key="p.project.property.projectref.add">refProject</entry>
  <entry key="refProject.label">Reference Project Setting</entry>
  <entry key="refProject.selectfrom">projectType1</entry>

reource_ja.properties
  <entry key="prtemplate.label">参照するプロジェクトの指定</entry>
```

4.17. プロジェクトのプロパティにプラグイン固有の任意データを追加

Nirvana のプロジェクトのプロパティ情報にプラグイン固有の任意データを追加できます. この情報は Nirvana のプロジェクトデータとして永続化されます. また, その情報の編集画面をプロジェクトのプロパティダイアログに追加できます. プロジェクトのプロパティ情報にプラグイン固有の任意データを追加するには, クラスの実装が必要です. また, プロジェクトのプロパティダイアログに設定パネルを追加するには, リソースの定義とクラスの実装が必要です. リソースの定義と, クラスの実装を順に説明します.

4.17.1.リソースの定義

プロジェクトのプロパティダイアログに編集画面を追加するには, キー「p.projectProperty.add」に値として, プラグインパッケージ名.gui パッケージに配置した, 設定パネルのクラス名を指定します. このクラスの詳細は 4.17.2 項を参照して下さい.

リソースファイルに以下の記述を行います.

(1) キー「p.projectProperty.add」の値に, リソースファイル内で一意となるキー名として, 設定パネルのクラス名を指定します.
(2) キー「設定パネルのクラス名」+「.label」の値に, Nirvana の環境設定ダイアログに表示されるタブ名を指定します.

例) 設定パネルのクラス名を「PluginPeojectPropertySetupPanel」, Nirvana のプロジェクトのプロパティダイアログに表示されるタブ名を「NewMethod Project Setup」, 日本語環境では「新表記法プロジェクト設定」と指定する場合

```
reource.properties
  <entry key="p.projectProperty.add">PluginProjectPropertyPanel</entry>
  <entry key="PluginProjectPropertyPanel.label">NewMethod Project Setup</entry>

reource_ja.properties
  <entry key="PluginProjectPropertyPanel.label">新表記法プロジェクト設定</entry>
```

4.17.2.クラスの実装

プロジェクトのプロパティ情報にプラグイン固有の任意データを格納, 取得するには, 以下のメソッドを呼び出します.

jp.co.nil.nirvana.model.Project#setPluginData(String pluginPackageName, String data)
jp.co.nil.nirvana.model.Project#getPluginData(String pluginPackageName)

なお, Project のインスタンスは jp.co.nil.nirvana.pluginapi.model.PVersionedDiagraml#getProject() から取得出来ます.

プロジェクトのプロパティダイアログに設定パネルを追加するには,
jp.co.nil.nirvana.pluginapi.gui.CustomProjectPropertyPanel を継承したクラスを, プラグインパッケージ名.gui パッケージに作成します.