IBM ILOG Views

Grapher V5.3

ユーザ・マニュアル

2009 年 **6** 月

著作権の告知

商標

IBM、IBM ロゴ、ibm.com、Websphere、ILOG、ILOG のデザイン、および CPLEX は、世界中の多くの国の管轄権で登録されている International Business Machines Corp. の商標または登録商標です。その他の製品およびサービス名は、IBM またはその他の企業の商標です。IBM 社の現在の商標一覧は、http://www.ibm.com/legal/copytrade.shtml にある Copyright and trademark information (著作権と商標についての情報) にあります。

Adobe、Adobe のロゴ、PostScript、および PostScript のロゴは、米国およびその他の国における Adobe Systems Incorporated の商標または登録商標です。

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

Microsoft、Windows、Windows NT、および Windows のロゴは、米国およびその他の国における Microsoft Corporation の商標です。

Java およびすべての Java に基づいた商標とロゴは、米国およびその他の国の Sun Microsystems, Inc. の商標です。

その他の企業、製品およびサービス名は、その他の企業の商標またはサービス商標です。

告知

詳細は、インストールした製品の <*installdir*>/license/notices.txt を参照してください。

目次

本書について

本*ユーザ・マニュアル*では、IBM® ILOG® Views の高度なパッケージであるグラファーについて説明します。

前提事項

本書では、特定のウィンドウシステムを含め、ユーザが IBM® ILOG® Views を使用する PC や UNIX® 環境について精通していることが前提となっています。IBM ILOG Views は C++ 開発者用に作成されているため、このマニュアルでは、ユーザが C++ のコードを作成できること、および C++ の開発環境について精通しており、ファイルやディレクトリの操作、テキスト・エディタの使用、C++ プログラムのコンパイルおよび実行ができることも前提となっています。

マニュアル構成

このマニュアルは、以下の章で構成されています。

- ***IBM ILOG Views Studio の Grapher 拡張機能の概要***では、IBM® ILOG® Views Studio でグラファー拡張機能を使う方法を説明します。

- ***グラファー・パッケージの機能***では、階層情報や相互関連情報のグラフィック表示を専門に行う機能について説明します。

表記法

書体の規則

以下の書体に関する規則は、このマニュアル全体に適用されます。

- コードの引用やファイル名は `courier` 書体で記載されます。
- ユーザが入力する項目は、*`courier`* 書体で記載されます。
- 初出の*斜体*の用語には、ユーザ・マニュアルの用語集で解説されているものがあります。

命名規則

以下の命名規則は、マニュアル全体を通して API に適用されます。

- IBM ILOG Views Foundation ライブラリで定義されている型、クラス、関数、マクロの名前は `Ilv` で始まります。
- クラス名、およびグローバル関数は、最初の文字が大文字で表された連結語として記載されます。

```
class IlvDrawingView;
```

- 仮想および通常メソッドの名前は小文字で始まります。スタティック・メソッドの名前は大文字で始まります。例：

```
virtual IlvClassInfo* getClassInfo() const;

static IlvClassInfo* ClassInfo*() const;
```

章

1

IBM ILOG Views Studio の Grapher 拡張機能の概要

この章では、IBM® ILOG® Views Studio の Grapher 拡張機能を紹介します。以下のトピックに関する情報が記載されています。

- メイン・ウィンドウ
- パレット・パネル
- グラファー拡張コマンド

> ***メモ:*** *IBM ILOG Views の Grapher 拡張機能の使用に関する章では、ユーザが IBM ILOG Views Studio ユーザ・マニュアルに記載されている内容に精通していることを前提としています。*

メイン・ウィンドウ

アプリケーションを起動させると、次のように、IBM® ILOG® Views Studio のメイン・ウィンドウが開きます。

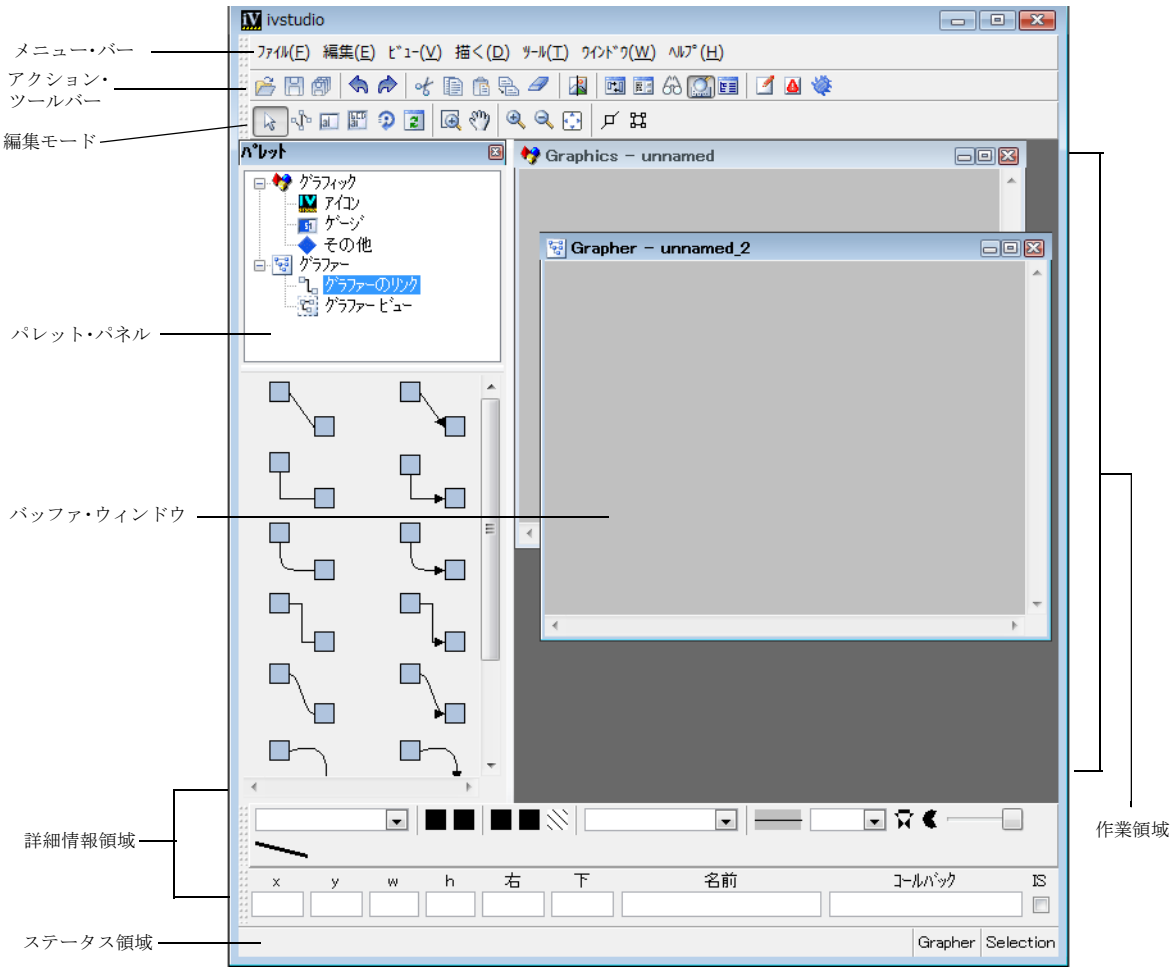

図1.1 *グラファー拡張機能のある* IBM ILOG Views *Studio メイン・ウィンドウ（起動時）*

メイン・ウィンドウは、Foundations パッケージだけがインストールされている場合とあまり変わりません。ただし、グラファー・パッケージがあると、追加のバッファ・ウィンドウが使用でき、パレット・パネルにパレットが追加され、インターフェースのメニュー・バーとツールバーにもアイテムが追加されます。

バッファ・ウィンドウ

アプリケーションおよびパネルは、メイン・ウィンドウに表示されているバッファ・ウィンドウに作成されます。カレント・バッファ・タイプはメイン・ウィンドウの一番下に表示されます。

IBM® ILOG® Views Studio の Grapher 拡張機能では、バッファの次のタイプを編集することができます。

- グラファー
- 2D グラフィック

IBM ILOG Views Studio を起動すると、空のグラフィック・バッファがデフォルトで表示されます。

> ***メモ:*** *メイン・ウィンドウで別のタイプのバッファに切り替えると、次のような違いがあることがわかります。*
>
> *それぞれのバッファ・タイプには独自の編集モード・セットがあります。カレント・バッファを変更するとき、ツールバーのアイコンとして利用可能な編集モードも変更します。*

Grapher バッファ・ウィンドウ

Grapher バッファ・ウィンドウを使うと、グラフを表示および編集できます。`IlvGrapher` を使用して、ノードとリンクの読み込み、編集、保存ができます。

新規の Grapher バッファ・ウィンドウを作成する方法は次の通りです。

1. [ファイル] メニューから [新規] を選択します。
2. 次に、表示されるサブメニューで [グラファー] を選択します。

このウィンドウを開くには、[ツール] メニューから [コマンド] を選択して表示する [コマンド] パネルから `NewGrapherBuffer` コマンドを実行することもできます。

`IlvGrapher` によって生成された `.ilv` ファイルを開くと、Grapher バッファ・ウィンドウが自動的に開きます。

2D Graphics バッファ・ウィンドウ

2D グラフィック・バッファは、Foundation パッケージのデフォルトです。IBM ILOG Views Studio の Grapher 拡張機能で引き続き利用できます。これにより `IlvManager` あるいは `IlvContainer` の内容を編集できます。オブジェクトの読み込み、編集、保存に `IlvManager` を使用します。

新規 2D Graphics バッファ・ウィンドウを作成するには、次の処理を行います。

1. [ファイル] メニューから [新規] を選択します。
2. 表示されたサブメニューで [2D グラフィック] を選択します。

このウィンドウを開くには、[ツール] メニューから [コマンド] を選択して表示する [コマンド] パネルから `NewGraphicBuffer` コマンドを実行することもできます。

IlvManager によって生成された .ilv ファイルを開くと、2D Graphics バッファ・ウィンドウが自動的に開きます。

メニュー・バー

グラファー・パッケージをインストールすると、メイン・ウィンドウのメニュー・バーで追加コマンドを使用できるようになります。

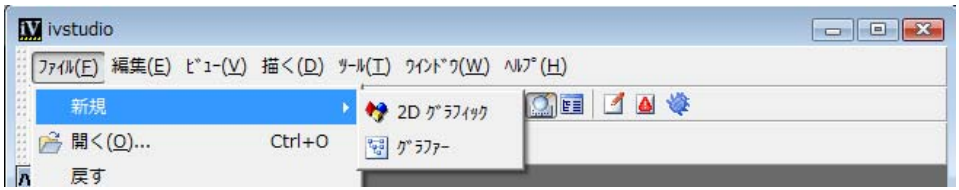

図 1.2 *IBM ILOG Views Studio Grapher 拡張機能メニュー・バー*

[ファイル] > [新規] メニューに、メニュー・アイテム [グラファー] が加わっています。これで新規のグラファー・バッファを作成できます。これはコマンド NewGrapherBuffer です。

アクション・ツールバー

アクション・ツールバーは Foundation パッケージと同じです。

編集モード・ツールバー

グラファー・バッファが作業領域のアクティブ・ウィンドウになっている場合、編集モード・ツールバーが次のように表示されます。

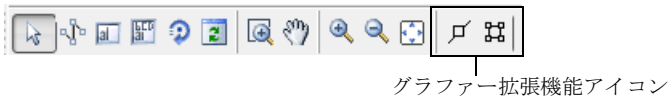

図 1.3 *IBM ILOG Views Studio グラファー拡張機能の編集モード・ツールバー*

ノード - 選択したオブジェクトをノードにする場合、このボタンを使用します。MakeNode コマンドを実装します。

ピン編集モード - グラファー・ノードに定義された接続ピンを対話的に編集するには、このモードを使用します。このモードの使用方法の詳細については、接続ピンの編集 を参照してください。

パレット・パネル

IBM® ILOG® Views Studio のグラファー拡張機能を使用するときは、パレット・パネルからグラファー・リンクへアクセスできます。

パレット・パネルの上部ペインに、グラファー拡張機能のあるパレットが 2 つ追加されていることが分かります。下部ペインにさまざまなグラファー・リンクを表示するには、上部ペインで適切なパレットをクリックしてください。

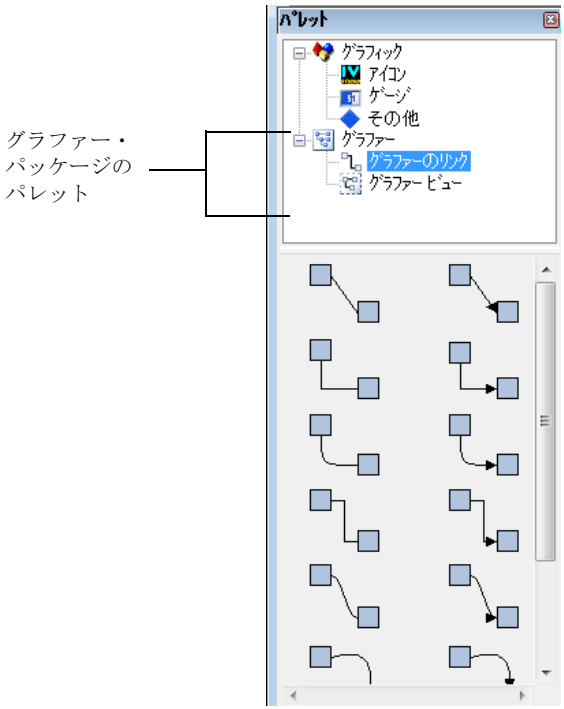

図 1.4 *IBM ILOG Views Studio グラファー拡張機能パレット・パネル*

以下のセクションでは、グラファー拡張機能で提供されているオブジェクトについて説明します。Foundation パッケージで提供されているオブジェクトの詳細は、IBM ILOG Views *Studio ユーザ・マニュアル*を参照してください。

グラファー・パレット

グラファー・パレットには、グラファー・リンクを作成するために使用できる以下のオブジェクトが含まれています (リンクは、コマンド・パネルでリンク編集コマンドを使用して作成することもできます)。リンク・モードを 選択 するには、2 つの `IlvShadowRectangle` の間のリンクをクリックします。これで、オレンジ色のボックスで囲まれたリンクが表示されます。

これらのモードはグラファー・バッファでのみ使用できます。

> ***メモ:*** *グラファー・リンクはノード間にしか作成できません。そのため、リンクするオブジェクトは、まず* `MakeNode` *コマンドでノードとして宣言する必要があります。オブジェクトを選択してから、[編集モード] ツールバーで [ノード] ボタンをクリックします。*

ArcLinkImage

`IlvArcLinkImage` オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

DoubleLinkImage

`IlvDoubleLinkImage` オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

DoubleSplineLinkImage

`IlvDoubleSplineLinkImage` オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

LinkImage

`IlvLinkImage` オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OneLinkImage

IlvOneLinkImage オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OneSplineLinkImage

IlvOneSplineLinkImage オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OrientedArcLinkImage

IlvArcLinkImage オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OrientedDoubleLinkImage

IlvDoubleLinkImage オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OrientedDoubleSplineLinkImage

IlvDoubleSplineLinkImage オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OrientedLinkImage

IlvLinkImage オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OrientedOneLinkImage

IlvOneLinkImage オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OrientedOneSplineLinkImage

`IlvOneSplineLinkImage` オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードの上で左クリックし、カーソルを 2 つ目のノードにドラッグします。マウスを放すと操作が終了します。

OrientedPolylineLinkImage

`IlvPolylineLinkImage` オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードをクリックし、必要に応じて中間点もクリックし、2 つ目のノードをダブルクリックして操作を終了します。

PolylineLinkImage

`IlvPolylineLinkImage` オブジェクトの 2 つのグラファー・ノードをリンクするには、このモードを使用します。1 つ目のノードをクリックし、必要に応じて中間点もクリックし、2 つ目のノードをダブルクリックして操作を終了します。

IlvSCGrapherRectangle

これは `IlvGrapher` の内容を表示するために `IlvSCGrapherRectangle` オブジェクトを作成します。ドラッグ・アンド・ドロップ操作または作成モード操作のいずれかを使用します (このコマンドはグラファー・ビュー・パレットにあります)。

グラファー拡張コマンド

このセクションでは、IBM® ILOG® Views Studio のグラファー拡張機能で使用可能な追加の定義済みコマンドの一覧表を、アルファベット順に表示します (IBM ILOG Views Studio Foundation コマンドもすべて使用できます)。一覧表には、各コマンドのラベル、コマンド・パネル以外からアクセスできる場合はそのアクセス方法、コマンドが属するカテゴリ、およびコマンドの用途を列挙しています。

コマンド・パネルを表示するには、メイン・ウィンドウの [ツール] メニューから [コマンド] を選択するか、[アクション] ツールバーの [コマンド] アイコン  をクリックします。

MakeNode

ラベル	ノード
パス	メイン・ウィンドウ : グラファー・バッファを編集している場合は [編集モード] ツールバー
カテゴリ	グラファー、スタジオ
アクション	カレント・バッファがグラファー・バッファである場合、このコマンドは選択したオブジェクトをノードにします。

NewGrapherBuffer

ラベル	グラファー
パス	メイン・ウィンドウ : [ファイル] メニュー > 新規
カテゴリ	バッファ、グラファー
アクション	新規グラファー・バッファの作成 このバッファが、カレント・バッファになります。

SelectArcLinkImageMode

ラベル	Arc-shaped link
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ円弧形のリンクを作成します。*IlvArcLinkImage* を参照してください。

SelectDoubleLinkImageMode

ラベル	DoubleLinkImage
パス	パレット・パネル : グラファー・リンク・パレット

カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ 2 ヶ所が曲がったリンクを作成します。 *IlvDoubleLinkImage* を参照してください。

SelectDoubleSplineLinkImageMode

ラベル	DoubleSplineLinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ 2 ヶ所が曲がった曲線リンクを作成します。 *IlvDoubleSplineLinkImage* を参照してください。

SelectLinkImageMode

ラベル	LinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのリンク間にダイレクトなリンクを作成します。*リンクのベース・クラス* を参照してください。

SelectOneLinkImageMode

ラベル	OneLinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ 1 ヶ所が曲がったリンクを作成します。 *IlvOneLinkImage* を参照してください。

SelectOneSplineLinkImageMode

ラベル	OneSplineLinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ 1 ヶ所が曲がった曲線リンクを作成します。*IlvOneSplineLinkImage* を参照してください。

SelectOrientedArcLinkImageMode

ラベル	Oriented Arc-shaped link
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ、有向円弧形のリンクを作成します。*IlvArcLinkImage* を参照してください。

SelectOrientedDoubleLinkImageMode

ラベル	Oriented DoubleLinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ、有向の 2 ヶ所が曲がったリンクを作成します。*IlvDoubleLinkImage* を参照してください。

SelectOrientedDoubleSplineLinkImageMode

ラベル	Oriented DoubleSplineLinkImage
パス	パレット・パネル : グラファー・リンク・パレット

カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ、有向の 2 ヶ所が曲がった曲線リンクを作成します。*IlvDoubleSplineLinkImage* を参照してください。

SelectOrientedLinkImageMode

ラベル	Oriented LinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ、有向のダイレクトなリンクを作成します *リンクのベース・クラス* を参照してください。

SelectOrientedOneLinkImageMode

ラベル	Oriented OneLinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ、有向の 1 ヶ所が曲がったリンクを作成します。*IlvOneLinkImage* を参照してください。

SelectOrientedOneSplineLinkImageMode

ラベル	Oriented OneSplineLinkImage
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ、有向の 1 ヶ所が曲がった曲線リンクを作成します。*IlvOneSplineLinkImage* を参照してください。

SelectOrientedPolylineLinkImageMode

ラベル	Free-shape oriented link
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ、有向の自由な形状のリンクを作成します。*IlvPolylineLinkImage* を参照してください。

SelectPinEditorMode

ラベル	PinEditor
パス	メイン・ウィンドウ : グラファー・バッファを編集している場合は [編集モード] ツールバー
カテゴリ	グラファ
アクション	カレント・バッファをピン編集モードに設定します。接続ピンの編集 を参照してください。

SelectPolylineLinkImageMode

ラベル	Free-shape link
パス	パレット・パネル : グラファー・リンク・パレット
カテゴリ	モード、グラファー
アクション	2 つのノードを結ぶ自由な形状のリンクを作成します。*IlvPolylineLinkImage* を参照してください。

章

2

グラファー・パッケージの機能

このセクションでは、IBM® ILOG® Views の高度なパッケージであるグラファーについて説明します。このパッケージは、階層情報や相互関連情報のグラフィック表示を専門に行う機能を提供します。このセクションは、次のような構成になっています。

- *グラフ管理* - 最初のセクションでは、グラフ管理クラス `IlvGrapher` について説明します。このクラスは、マネージャの概念を自然に拡張したものです。これは `IlvManager` クラスに基づいており、相互接続されたグラフィック・オブジェクトを処理する組み込み機構を提供します。
- *グラファーのリンク* - 2 番目のセクションでは*グラファー・リンク*の概念を説明し、カスタマイズ可能なグラフィック・オブジェクトのクラス階層がそれらのエンティティをどのように表現するかを示します。
- *グラファー・インタラクタ* - 3 番目のセクションは、複数のインタラクタ・ファミリーを使用してどのようにグラフ表現を操作できるかを説明します。

グラフ管理

このセクションでは、IBM® ILOG® Views でのグラフ管理について説明します。説明は、次の 2 つの部分に分かれています。

- *IlvGrapher クラスの説明*

IlvGrapher クラスの説明

グラフを表現するグラフィック・オブジェクトは IlvGrapher クラスのインスタンスに格納されます。このクラスは IlvManager クラスから派生し、その機能をすべて継承します。IlvManager (ベース・クラス) のコンストラクタと IlvGrapher のコンストラクタには同じパラメータがあります。

```
IlvGrapher(IlvDisplay*     display,
           int             layers    =  2,
           IlvBoolean      useacc    =  IlvTrue,
           IlvUShort       maxInList =  IlvMaxObjectsInList,
           IlvUShort       maxInNode =  IlvMaxObjectsInList);
```

IlvManager の概念に加え、IlvGrapher クラスでは次の 3 種類のグラフィック・オブジェクトを区別できます。

- **ノード** - ノードは情報階層の中の視覚的な参照点です。ノードは、IlvGrapher::addNode メソッドによりグラファーに追加されると特定の機能を持つグラフィック・オブジェクトで、IlvGraphic クラスのサブタイプです。この機能によって、ノードの移動時にも、リンクとノードの結び付きは解除されません。
- **リンク** - リンクは、ノード間の接続の視覚的な表現です。リンクは、IlvLinkImage クラスまたはそのいずれかのサブクラスのインスタンスです。これは IlvGrapher::addLink メソッドでグラファーに追加されます。リンクは既存の 2 ノード間でのみ存在できるため、グラファーでノードと呼ばれている 2 つのグラフィック・オブジェクトとともに作成する必要があります。*ゴースト・ノード*(IlvGrapher::addGhostNode メソッドで追加される) を使用すると、端が接続されていないリンクを作成できます。
- **一般グラフィック・オブジェクト** - 基本の IlvManager インスタンスの場合と同様に、ノードまたはリンク以外のあらゆる IlvGraphic オブジェクトをグラフに組み込むことができます。

IlvGrapher クラスは、リンクとノードを管理するメンバ関数のセットを提供します。たとえば、IlvGrapher::changeLink メソッドを呼び出すことにより、あるリンクを他のリンクに置き換えることができます。

また、IlvGrapher::makeNode メソッドを呼び出すことにより、グラファーに格納されているグラフィック・オブジェクトをノードに変換できます。このメソッドをグラファー・リンクに適用できます。これにより、リンクを他のノードに接続できます。ノードの振る舞いを持つリンクを処理する場合は、このリンクの位置を制御する幾何学的依存関係に閉路がないことを確認する必要があります。同様に、IlvGrapher::makeLink メソッドを使用してグラフィック・オブジェクトをグラファーのリンクに変換できます。作成されたリンクは IlvLinkHandle クラスのインスタンスになります。説明は*グラファーのリンク*を参照してください。

オブジェクトを IlvGrapher に格納すると、IlvGrapher::isNode メソッドと IlvGrapher::isLink メソッドを使用してノード、リンクおよび通常のグラフィック・インスタンスを区別できるようになります。

IlvGrapher API は、グラフの位相を問い合わせる複数のメソッドも提供します。たとえば、IlvGrapher::isLinkBetween メソッドを使用することにより、指定された 2 つのノードが接続されているかどうかを確認できます。また、IlvGrapher::getLinks メソッドを使用することにより、ノードのすべての出発リンクまたは到着リンクを取得することもできます。

以下のサンプル・コードは、::mapLinksIlvGrapher::mapLinks メソッドを使用してノードの出発リンクをすべて選択する方法を示します。

```
static void SelectLink(IlvGraphic* g, IlvAny arg)
{
ILVCAST(IlvGrapher*,arg)->setSelected(g,IlvTrue);
}

{
...
IlvGrapher* graph = ....;
IlvGraphic* node = ....; // The node being considered
//== Call the SelectLink function on all outgoing links of <node>
graph->mapLinks(node,SelectLink,graph,IlvLinkFrom);
...
}
```

最後に、IlvGrapher クラスはノードを垂直または水平なツリー構造に再編成する 2 つの定義済みレイアウト・メソッドを提供します。これらのレイアウトは、IlvGrapher::nodeXPretty メソッドと IlvGrapher::nodeYPretty メソッドで実装されています。

シンプルなグラファーの作成方法を示す例が、<ILVHOME>/samples/grapher/simple ディレクトリにあります。また、IlvGrapher クラスのメンバ関数の詳細については、*IBM ILOG Views Grapher リファレンス・マニュアル*を参照してください。

グラフ記述の読み込みと保存

IlvGrapher クラスは IlvGraphInputFile クラスを使用してグラフを読み込み、IlvGraphOutputFile クラスを使用してグラフを保存します。

IlvGraphOutputFile

IlvGraphOutputFile クラスは IlvManagerOutputFile のサブクラスです。このサブクラスでは、各オブジェクトの記述ブロックの前にそのオブジェクトの詳細情報を追加するために、仮想メソッド IlvGraphOutputFile::writeObject が再定義されています。ここでは、この情報は、レイヤ・インデックス、オブジェクトのタイプ (ノード、リンク、またはその他のタイプのオブジェクト) および接続ピンです。接続ピンについては、*グラファーのリンク*に説明があります。

IlvGraphInputFile

IlvGraphInputFile クラスは IlvManagerInputFile のサブクラスです。このサブクラスでは、IlvGraphOutputFile::writeObject メソッドに記述された詳細情報を読み込むために、仮想メソッド IlvGraphInputFile::readObject が再定義されています。

グラファーのリンク

このセクションでは、グラファーにリンクを実装する C++ クラスを説明します。これらのクラスは IlvGraphic クラスのインターフェースを継承しており、リンクとそれに接続されたノードの関係を処理する特定のメソッド追加します。以下の項目について説明します。

- *リンクのベース・クラス*
- *定義済みグラファー・リンク*
- *カスタム・グラファー・リンクの作成*
- *接続ピン*

リンクのベース・クラス

図 2.1 は、2 つのノードをつなぐ直線リンクを示します。

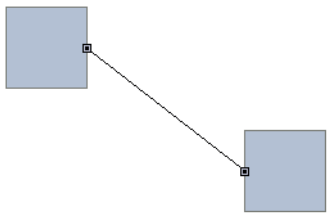

図 2.1 *2 つのノード間の直接リンク*

IlvLinkImage インスタンスは 2 つのノード間の接続を表すグラフィック・オブジェクトです。デフォルトでは、2 つのノードを結合する直線で表現されます。IlvLinkImage クラスのコンストラクタは以下の通りです。

```
IlvLinkImage(IlvDisplay* display,
             IlvBoolean  oriented,
             IlvGraphic* from,
             IlvGraphic* to,
             IlvPalette* palette=0);
```

from パラメータは、リンクの開始ノードを示す IlvGraphic タイプのオブジェクトです。to パラメータは、リンクの終了ノードを示す IlvGraphic タイプのオブジェクトです。oriented パラメータは、リンクが矢印で終わるかどうかを示します。

前に set と get の付いている複数のメンバ関数を使用すると、これらのプロパティにアクセスできます。たとえば、終了ノードは IlvLinkImage::getTo メソッドと IlvLinkImage::setTo メソッドでアクセスできます。同様に、IlvLinkImage::setOriented メソッドを使用してリンクの有向モードを変更することができます。

これらのプロパティの格納に加え、IlvLinkImage クラスには次の目的もあります。

- リンクの形状を関連するノードの関数として計算し、ノードのジオメトリが変化した場合のリンクの振る舞いを定義します。このタスクは、IlvLinkImage::getLinkPoints 仮想メソッドによって実行されます。
- リンクがどのように描画されるかを定義します。これは、計算された形状を使用して行われ、IlvGraphic クラスから継承された仮想メソッドに実装されます。

異なる振る舞いおよび / または描画アスペクトを持つリンクを作成する場合は、IlvLinkImage をサブクラス化すると役に立ちます。振る舞いを変更するには、IlvLinkImage::getLinkPoints メソッドをオーバーライドします。

```
virtual IlvPoint* getLinkPoints(IlvUInt& count,
                                const IlvTransformer* t) const;
```

返された配列を呼び出し側で削除することはできません。この配列を IlvPointPool クラスを使用して共通のメモリ・プールに割り当てる必要があります。このメソッドでは、開始ノードと終了ノードのジオメトリを問い合わせてリンクの形状を定義する点を決めることができます。このような点には 2 つのカテゴリがあります。

- リンクの終点。これらはリンクが開始する位置と終了する位置を定義します。
- 中間点。これらはリンクの全体的なアスペクトを定義します。

IlvLinkImage クラスは、リンクの終点位置の計算に IlvLinkImage::computePoints メソッドを使用します。

```
virtual void computePoints(IlvPoint& src,
                           IlvPoint& dst,
                           const IlvTransformer* t = 0) const;
```

デフォルトの実装では、まずリンクがノードの接続ピンに関連付けられているかどうかをチェックします (詳細については、*接続ピン管理クラス* を参照してください)。接続ピンが定義されていない場合、開始ノードと終了ノードのバウンディ

ング・ボックスとリンクの交差が計算されます。図 2.2 は、この関係を図示したものです。

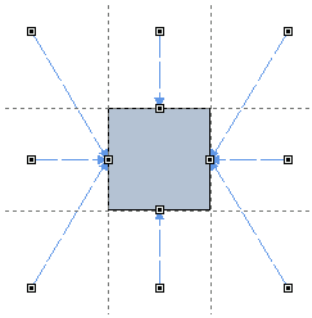

図2.2　接続ピンが定義されていない場合の終点の位置

定義済みグラファー・リンク

定義済みのリンク・クラスはグラファー・ライブラリにあります。これらのクラスはそれぞれ、`IlvLinkImage` ベース・クラスに特定の振る舞いや描画機能を追加します。これらのクラスをそのまま使用したり、サブクラス化してカスタマイズされたリンクを作成できます。以下のクラスを使用できます。

- *IlvLinkHandle*
- *IlvLinkLabel*
- *IlvOneLinkImage*
- *IlvOneSplineLinkImage*
- *IlvDoubleLinkImage*
- *IlvDoubleSplineLinkImage*
- *IlvArcLinkImage*
- *IlvPolylineLinkImage*

IlvLinkHandle

`IlvLinkHandle` クラスはリンク・クラスの例で、リンクの形状と振る舞いは `IlvLinkImage` から直接継承され、リンクの描画のみが再定義されています。

このクラスにより、あらゆるタイプのグラフィック・オブジェクトを参照して、グラファー・リンクとして振る舞うようにできます。また、グラフィック・オブジェクトは複数の `IlvLinkHandle` インスタンスから参照できます。これにより、形状が複雑でも非常に軽量なリンクを作成することができます。図 2.3 は、多角形を参照する `IlvLinkHandle` インスタンスの例を示します。

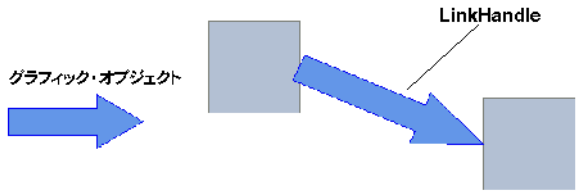

図2.3　リンクとして使用されるグラフィック・オブジェクト

このクラスのコンストラクタは以下の通りです。

```
IlvLinkHandle(IlvDisplay* display,
              IlvGraphic* object,
              IlvGraphic* from,
              IlvGraphic* to,
              IlvDim width = 0,
              IlvBoolean owner = IlvTrue
              IlvPalette* palette=0);
```

グラファーに追加されると、このインスタンスは幅 `width` を使用してノード `from` と `to` の間のリンクとしてグラフィック・オブジェクト `object` を描きます。`owner` パラメータはハンドルと被参照オブジェクトの間の関係を示します。ハンドルが被参照オブジェクトを所有している場合、このオブジェクトの削除はハンドルが行います。つまり、いずれのハンドルにも所有されていない被参照オブジェクトは、安全に共有することができます。

`IlvLinkHandle` クラスの使用方法の例は、`<ILVHOME>/samples/grapher/linkhand` ディレクトリにあります。

IlvLinkLabel

`IlvLinkLabel` クラスはまた、`IlvLinkImage` クラスの形状と振る舞いを継承します。`IlvLinkLabel` タイプのリンクは、ユーザ定義の文字列でラベル付けされます。

この文字列は、コンストラクタの `label` パラメータによって指定できます。リンクの作成後に `IlvLinkLabel::setLabel` メソッドを使用して指定することもできます。

図 2.4 は、次の 2 つの `IlvLinkLabel` オブジェクトを示しています。

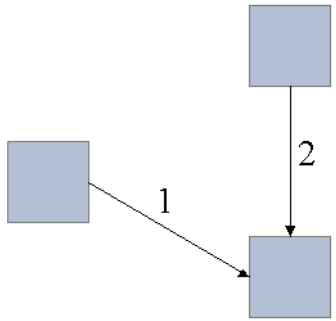

図 2.4　ラベル付けされたリンク

IlvOneLinkImage

`IlvOneLinkImage` クラスは、`IlvLinkImage` クラスから派生し、新しい形状と振る舞いを定義します。このクラスのインスタンスは 図 2.5 に図示されているとおり、2 本の直角線から構成されています。

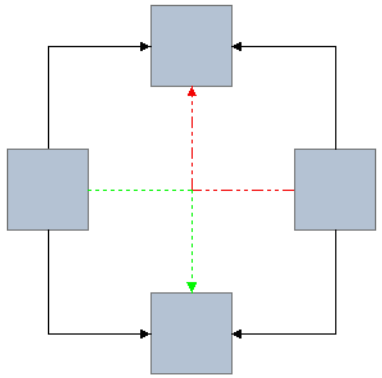

図 2.5　IlvOneLinkImage

リンクの形状は*向き*プロパティに応じて変わります。これは、`from` ノードを出発したリンクが垂直方向 (`IlvVerticalLink`) に進むか水平方向 (`IlvHorizontalLink`) に進むかを示します。このプロパティはコンストラクタで指定することも、リンク作成後に `IlvOneLinkImage::setOrientation` メソッドを使用して指定することもできます。

IlvOneSplineLinkImage

このクラスは、リンクをスプラインとして描画する `IlvOneLinkImage` のサブクラスです。

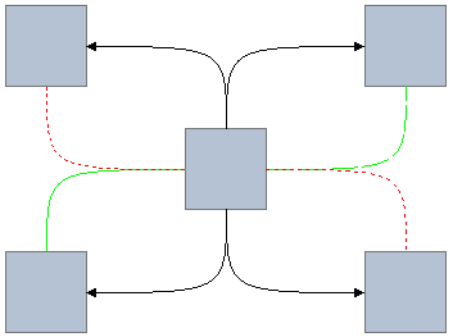

図 2.6　IlvOneSplineLinkImage

終点の位置は `IlvOneLinkImage` クラスで計算されたものと同じです。描画されたスプラインの 2 つの制御点は、両方ともリンクの開始接線と終了接線の交差点にあります。`IlvOneSplineLinkImage::setControlPoint` メソッドを使用することにより、二重制御点の位置を変更できます。

IlvDoubleLinkImage

`IlvDoubleLinkImage` クラスは、`IlvLinkImage` から派生し、新しい形状と振る舞いを定義します。このクラスのインスタンスは図 2.7 に図示されている通り、直角に交わる 3 本の一続きの線から構成されています。

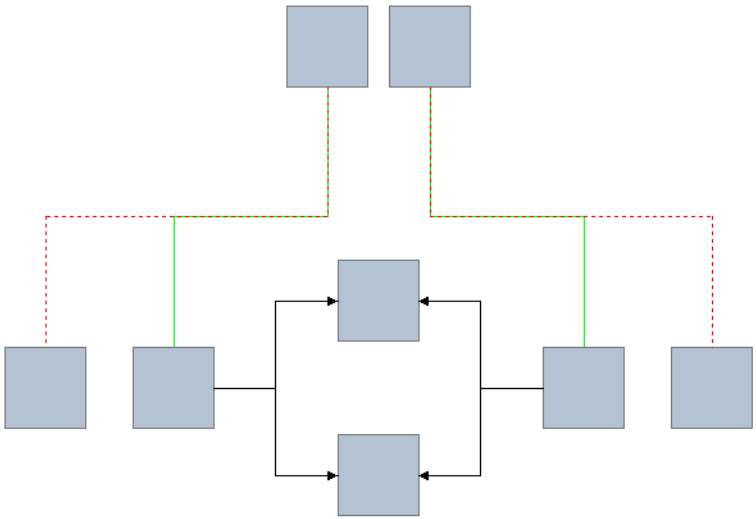

図 2.7　IlvDoubleLinkImage

3 つの切片のレイアウトは、IlvDoubleLinkImage::setFixedOrientation メソッドで設定された 2 つのモードに従います。

- 自動 - 切片の方向は、2 つのノードの間の水平と垂直の分離に依存します。切片の中間が最大分離の向きを取ります。
- 固定 - リンクの向きが固定され、開始ノードを離れるときのリンクの方向 (水平か垂直) を指定します。

IlvDoubleSplineLinkImage

IlvDoubleSplineLinkImage クラスは IlvDoubleLinkImage のサブクラスであり、図 2.8 で示すように線分の代わりに滑らかな曲線でリンクを描画します。これらのリンクの振る舞いは IlvDoubleLinkImage クラスの場合と同じです。

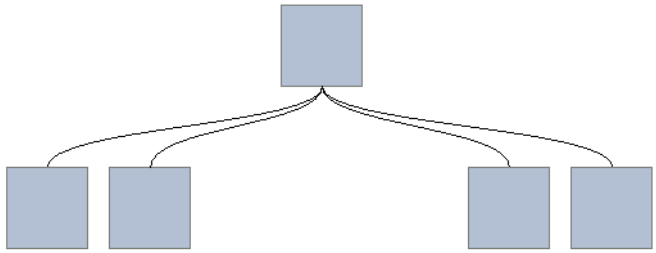

図 2.8 *IlvDoubleSplineLinkImage*

IlvArcLinkImage

IlvArcLinkImage クラスは、新しい形状と振る舞いを定義する IlvLinkImage のサブクラスです。このタイプのリンクは、図 2.9 で示すように 2 つのノードをつなぐ円弧として描画されます。

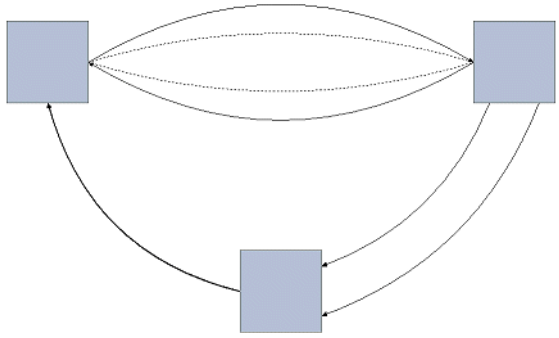

図 2.9 *3 つのノードをつなぐ IlvArcLinkImage*

円弧は 2 つ制御点のあるスプラインとして描画されます。これらの制御点とリンクの終点をつなぐ切片の間の距離 (*円弧のオフセット*とも呼ばれます) は、以下のいずれかで指定できます。

- 固定値。`IlvArcLinkImage::setFixedOffset` メソッドを使用。
- 切片の長さに比例する値。`IlvArcLinkImage::setOffsetRatio` メソッドを使用。

円弧オフセットはマイナスの値をとることもできます。この場合、制御点は開始点と終点をつなぐ有向切片の右側にあります。このため、異なる円弧オフセットを使用することにより、2 つのノードを重複なしに複数のリンクで接続することができます。

IlvPolylineLinkImage

このクラスを使うと、リンクの中間点を動的に定義することができます。これらの点は各 `IlvPolylineLinkImage` インスタンスに格納され、複数の方法で指定できます。

- `IlvPolylineLinkImage::setPoints`
- `IlvPolylineLinkImage::addPoints`
- `IlvPolylineLinkImage::removePoints`
- `IlvPolylineLinkImage::movePoint`

すべてのリンク・クラスの場合と同様に、生成される形状は `IlvPolylineLinkImage::getLinkPoints` メソッドで計算されます。また、リンクを直線の線分で描画するか、または `IlvPolylineLinkImage::drawSpline` メソッドを呼び出して曲線で描画するかを指定することもできます。図 2.10 は、`IlvPolylineLinkImage` インスタンスで作成される自由形式のリンクの例を示しています。

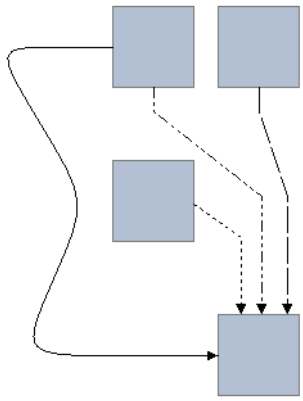

図 2.10　IlvPolylineLinkImage

カスタム・グラファー・リンクの作成

このセクションでは、以下の要件を満たすグラファー・リンクを作成するために IlvLinkImage をサブククラス化します。

- リンクは常に 2 つのノード間の直線として描画されます。
- 開始点は接続ピンで定義されているか、または開始ノードの中央にあります。
- 終点は、開始点にもっとも近い終了ノードの接面に対してリンクが直角に保たれる位置とします。これができなければ、終点はノード・バウンディング・ボックスのもっとも近い角に置かれます。

リンクはベース・クラス IlvLinkImage と同じ方法で描画されます。そのため、IlvGraphic から継承された関連メソッドは変更されずに残ります。また、リンクの形状を定義する点は 2 つしかありません (2 つの終点。中間点はなし)。リンク定義には 2 つの方法があります。IlvLinkImage::getLinkPoints メソッドまたは IlvLinkImage::computePoints メソッドをオーバーロードします。この例では 2 番目の方法を使っています。

```
void
MyLink::computePoints(IlvPoint& src,
                      IlvPoint& dst,
                      const IlvTransformer* t) const
{
    //== [1] ==
    IlvGrapherPin* pin = IlvGrapherPin::Get(getFrom());
    if (!pin || !pin->getLinkLocation(getFrom(),this,t,src)) {
        IlvRect bbox;
        getFrom()->boundingBox(bbox,t);
        src.move(bbox.centerx(),bbox.centery());
    }

    //== [2] ==
```

```
    IlvRect toBBox;
    getTo()->boundingBox(toBBox,t);
    if (src.x()<toBBox.x()) {
        if (src.y() < toBBox.y()) // Upper left quadrant
            dst.move(toBBox.x(),
                     toBBox.y());
        else if (src.y() >= toBBox.bottom()) // Lower left quadrant
            dst.move(toBBox.x(),
                     toBBox.y()+toBBox.h()-1);
        else // Left quadrant
            dst.move(toBBox.x(),
                     src.y());
    } else if (src.x()>=toBBox.right()) {

        if (src.y() < toBBox.y()) // Upper right quadrant
            dst.move(toBBox.x()+toBBox.w()-1,
                     toBBox.y());
        else if (src.y() >= toBBox.bottom()) // Lower right quadrant
            dst.move(toBBox.x()+toBBox.w()-1,
                     toBBox.y()+toBBox.h()-1);
        else // Right quadrant
            dst.move(toBBox.x()+toBBox.w()-1,
                     src.y());
    } else {
        if (src.y() < toBBox.y()) // Upper quadrant
            dst.move(src.x(),
                     toBBox.y());
        else if (src.y() >= toBBox.bottom()) // Lower quadrant
            dst.move(src.x(),
                     toBBox.y()+toBBox.h()-1);
        else // src inside toBBox
            dst.move(toBBox.centerx(),toBBox.centery());
    }
}
```

コードの最初の部分 (`[1]`) で、この開始ノードに定義されている接続ピンにリンクが付加されているかどうかの検証が行われます。付加されていない場合には、このノードのバウンディング・ボックスの中央を返します。

開始点の位置が計算されると、終了ノードのバウンディング・ボックスに対する開始点の位置が検証されます (`[2]`)。可能なパターンは 9 つあり (`toBBox` で定義される 8 つの四分円と、開始点が `toBBox` の中にある場合)、それぞれは独自の位置を定義します。

接続ピン

接続ピンにより、グラファー・ノードのリンク終点の正確な位置を制御できます。リンクが接続ピンに付加されると、接続点は開始ノードと終了ノードの相対的な位置を問わず、同じ場所に固定されます。

このセクションでは、次の項目を説明します。

- *接続ピン管理クラス*

- *多目的 IlvGrapherPin サブクラス*
- *IlvGrapherPin クラスの拡張*

接続ピン管理クラス

IlvGrapherPin 抽象クラスは、接続ピンの集合を処理するためのものです。1 つ目の目標は、リンクとピンの結合を維持することです。そのため、ピンはインデックスにより参照されます。IlvGrapherPin::setPinIndex メソッドにより、特定の接続ピンへリンクを接続できます。

```
IlvLinkImage* link = …;
//== Recover the IlvGrapherPin instance associated with the starting node
IlvGrapherPin* pin  =  IlvGrapherPin::Get(link->getFrom());
//== Connect the link to the pin whose index is 0
pin->setPinIndex(link,0,IlvTrue);
```

同様に、IlvGrapherPin::getPinIndex メソッドを使用して、リンクの付加されている接続ピンのインデックスを復元できます。

IlvGrapherPin クラスの 2 つ目の目的は、特定のノードで使用できる接続点の座標を問い合わせるインターフェースを提供することです。それぞれの具体的なサブクラスは、IlvGrapherPin::getCardinal メソッドと IlvGrapherPin::getLocation メソッドの実装を提供する必要があります。

```
virtual IlvUInt getCardinal(const IlvGraphic* node,
                            const IlvTransformer* t) const;
```

このメソッドは、トランスフォーマ t で表示されるときに指定されたノード node のインスタンスが処理する接続ピンの数を返します。

```
virtual IlvBoolean getLocation(IlvUInt pinIndex,
                               const IlvGraphic* node,
                               const IlvTransformer* t,
                               IlvPoint& where) const;
```

このメソッドは where パラメータに、トランスフォーマ t. で表示されるときにノード node のインデックス pinIndex で指定される接続ピンの座標を返します。

このインターフェースのその他のメソッド (IlvGrapherPin::getClosest、IlvGrapherPin::getLinkLocation など) には、オーバーロード可能なデフォルト実装があります。たとえば、getClosest メソッドは使用可能なすべての接続ピンを考慮し、getLocation メソッドを使用します。このメソッドを次のように変更できます。

- より迅速に実装を提供する (getLocation には、getClosest でしか実行できない演算が含まれる可能性があります)。
- もっとも距離が近いものではなく最初の未使用のピンを返す。

多目的 IlvGrapherPin サブクラス

IlvGenericPin クラスは IlvGrapherPin の定義済みの具体的なサブクラスで、これにより接続ピンをノードに動的に定義できます。新しい接続ピンは、特定のトランスフォーマによって表示されるときに、ノード上の希望位置に指定されます。この位置が格納されると、IlvGenericPin クラスは適用されたトランスフォーマとは関係なく、接続ピンを正確に位置決めするためにオブジェクトの形状を使用します。

このクラスを使用してノードのバウンディング・ボックスの 4 隅に接続ピンを追加する方法の例は以下の通りです。

```
IlvGraphic* node = ...;
//== Create an empty instance of IlvGenericPin
IlvGenericPin* pin = new IlvGenericPin();
//== Add the four connecting points
IlvRect bbox;
node->boundingBox(bbox,0);
pin->addPin(node,IlvPoint(bbox.x(),bbox.y()),0);
pin->addPin(node,IlvPoint(bbox.x()+bbox.w()-1,bbox.y()),0);
pin->addPin(node,IlvPoint(bbox.x()+bbox.w()-1,bbox.y()+bbox.h()-1),0);
pin->addPin(node,IlvPoint(bbox.x(),bbox.y()+bbox.h()-1),0);
//== Attach the IlvGenericPin instance to the node
pin->set(node);
```

> ***メモ:*** *この例では、トランスフォーマが適用されないときにオブジェクト座標系に点が指定されます。*

IlvGrapherPin クラスの拡張

ここでは、ノードのバウンディング・ボックスの中央で単一の接続ピンを処理する、具体的な IlvGrapherPin サブクラスの例を示します。この CenterPin と呼ばれるクラスは、以下のように宣言されます。

```
#include <ilviews/grapher/pin.h>

class CenterPin
: public IlvGrapherPin
{
public:
    CenterPin() {}

    virtual IlvUInt getCardinal(const IlvGraphic*,
                                const IlvTransformer*) const;

    virtual IlvBoolean getLocation(IlvUInt,
                                   const IlvGraphic*,
                                   const IlvTransformer* t,
                                   IlvPoint&) const;
    DeclarePropertyInfoRO();
    DeclarePropertyIOConstructors(CenterPin);
};
```

CenterPin クラスは情報を一切格納しないため、このクラスのコンストラクタは何も行いません。CenterPin クラスの永続化には、DeclarePropertyInfoRO マクロと DeclarePropertyIOConstructors マクロが使用されます。他の IlvGrapherPin メソッドの実装は変更の必要がないため、getCardinal メソッドと getLocation メソッドのみがオーバーロードされます。CenterPin クラスのソース・ファイルは以下のメソッドを定義します。

```
#include <centerpin.h>

// --------------------------------------------------------------------------
// - IO Constructors
CenterPin::CenterPin(IlvInputFile& input, IlvSymbol* s)
: IlvGrapherPin(input, s) {}

CenterPin::CenterPin(const CenterPin& src)
: IlvGrapherPin(src) {}
// --------------------------------------------------------------------------
IlvUInt
CenterPin::getCardinal(const IlvGraphic*,
                       const IlvTransformer*) const
{
    return 1;
}

// --------------------------------------------------------------------------
IlvBoolean
CenterPin::getLocation(IlvUInt,
                       const IlvGraphic* node,
                       const IlvTransformer* t,
                       IlvPoint& where) const
{
    IlvRect bbox;
    node->boundingBox(bbox, t);
    where.move(bbox.centerx(), bbox.centery());
    return IlvTrue;
}

// --------------------------------------------------------------------------
// - Macros to register the class and make it persistent
IlvPredefinedPropertyIOMembers(CenterPin)
IlvRegisterPropertyClass(CenterPin, IlvGrapherPin);
```

getCardinal メソッドの実装は単純で、あらゆるノードとトランスフォーマに 1 を返します。getLocation メソッドは、ノードの変換済みバウンディング・ボックスを問い合わせ、その中心を返すだけです (このクラスはただ 1 つの接続ピンしか定義しないため、接続ピンのインデックスは使用されません)。CenterPin クラスの宣言は、ファイル <ILVHOME>/samples/grapher/include/centerpin.h にあります。この実装は、ファイル <ILVHOME>/samples/grapher/src/centerpin.cpp にあります。

グラファー・インタラクタ

IlvManager クラスは、オブジェクトを作成してその形状を変えるために使用する多種多様なインタラクタを提供します。IlvGrapher クラスには、新しいノードとリンクを作成してそれらの接続方法を変更するための特定インタラクタが含まれます。

- *選択インタラクタ*
- *ノードの作成*
- *リンクの作成*
- *接続ピンの編集*
- *リンクの編集*

選択インタラクタ

IlvGraphSelectInteractor クラスは IlvSelectInteractor クラスから派生しています。移動または拡大されるノードに付加されたリンクのゴースト・イメージの描画を管理するために使用する追加のメンバ関数が含まれています。このクラスには以下のコンストラクタがあります。

```
IlvGraphSelectInteractor(IlvManager* manager, IlvView* view);
```

このコンストラクタは、マネージャ manager に接続されたビュー view の個別オブジェクトまたはオブジェクトのグループの選択を可能にする IlvGraphSelectInteractor クラスの新しいインスタンスを初期化します。このマネージャは、IlvGrapher クラスのインスタンスであると想定されます。

ノードの作成

IlvMakeNodeInteractor クラスは、グラファーにノードをインタラクティブに作成できるインタラクタのベース・クラスです。このクラスのインスタンスは、ここで示すように、グラファーとそのいずれかの接続ビューに付加する必要があります。

```
IlvGrapher* graph = ...;
IlvView* view = ...;
IlvMakeNodeInteractor * inter = new IlvMakeNodeInteractor(graph, view);
graph->setInteractor(inter);
```

ノードを作成するには、作業中のビューで矩形領域をドラッグします。作成するグラフィック・オブジェクトの種類を指定する方法は 2 つあります。

- IlvMakeNodeInteractor クラスをサブタイプ化して、その IlvMakeNodeInteractor::createNode メソッドをオーバーロードする。

- IlvMakeNodeInteractorFactory クラスをサブタイプ化して、その IlvMakeNodeInteractorFactory::createNode メソッドをオーバーロードする。IlvMakeNodeInteractor::setFactory メソッドを使用して、ノード・ファクトリとインタラクタを関連付けることができます。

グラファー・ライブラリは、IlvMakeNodeInteractor の定義済みサブクラスを提供します。

- IlvMakeShadowNodeInteractor - このインタラクタは IlvShadowLabel クラスのインスタンスを作成してそれらをノードとしてグラファーに格納します。
- IlvMakeReliefNodeInteractor - このインタラクタは IlvReliefLabel クラスのインスタンスを作成してそれらをノードとしてグラファーに格納します。

リンクの作成

IlvMakeLinkInteractor クラスは、グラファーでノードをインタラクティブに接続できるインタラクタのベース・クラスです。コンストラクタは次の通りです。

```
IlvMakeLinkInteractor(IlvManager* manager,
                      IlvView* view,
                      IlvBoolean oriented = IlvTrue);
```

oriented パラメータは、作成されたリンクに向きがあるかどうかを指定します。以下に、このタイプのインタラクタを作成してグラファーとそのいずれかのビューに接続する方法の例を示します。

```
IlvGrapher* graph = ...;
IlvView* view = graph->getFirstView();
IlvMakeLinkInteractor * inter = new IlvMakeLinkInteractor(graph, view);
graph->setInteractor(inter);
```

2 つのノードを接続するには、次の手順に従います。

1. 開始ノードをクリックします。インタラクタがこのノードを有効であると判断した場合、ノードは強調表示されます。
2. マウスを終了ノードまでドラッグします。このノードが有効な場合は、同じように強調表示されます。
3. マウスを放すとリンクが作成されます。

IlvMakeLinkInteractor::acceptFrom メソッドと IlvMakeLinkInteractor::acceptTo メソッドをオーバーロードすることにより、どのノードを有効にするかを制御できます。作成するリンクの種類を指定する方法は、次の 2 つです。

- IlvMakeLinkInteractor クラスをサブタイプ化して、その IlvMakeLinkInteractor::createLink メソッドをオーバーロードする。

- IlvMakeLinkInteractorFactory クラスをサブタイプ化して、その IlvMakeLinkInteractorFactory::createLink メソッドをオーバーロードする。IlvMakeLinkInteractor::setFactory メソッドを使用して、リンク・ファクトリとインタラクタを関連付けることができます。

グラファー・ライブラリは、IlvMakeLinkInteractor の定義済みサブクラスを複数提供します。

- IlvMakeLinkImageInteractor - このクラスを使って、IlvLinkImage タイプのリンクを作成します。
- IlvMakeLabelLinkImageInteractor - このクラスを使って、IlvLinkLabel タイプのリンクを作成します。
- IlvMakeOneLinkImageInteractor - このクラスを使って、IlvOneLinkImage タイプのリンクを作成します。
- IlvMakeOneSplineLinkImageInteractor - このクラスを使って、IlvOneSplineLinkImage タイプのリンクを作成します。
- IlvMakeDoubleLinkImageInteractor - このクラスを使って、IlvDoubleLinkImage タイプのリンクを作成します。
- IlvMakeDoubleSplineLinkImageInteractor - このクラスを使って、IlvDoubleSplineLinkImage タイプのリンクを作成します。

ポリライン・リンクの作成

IlvMakePolyLinkInteractor クラスは、IlvMakeLinkInteractor から派生しない特殊なインタラクタです。

このインタラクタを使用して、中間点を明示的に定義できるリンクを作成します。これにより、IlvMakePolyLinkInteractor::accept メソッドを使用して描画できる形状を制御できます。

```
virtual IlvBoolean accept(IlvPoint& point);
```

このメソッドをオーバーロードすることにより、リンクの中間点の位置に特定の制約を加えることができます。これらの点を定義すると、リンクが IlvMakePolyLinkInteractor::makeLink メソッドで作成されます。このメソッドはサブクラスに定義し、適切なリンク・インスタンスを返すようにする必要があります。グラファー・ライブラリには定義済みサブクラスが 1 つあります (IlvMakePolylineLinkInteractor)。IlvPolylineLinkImage タイプのリンクを作成する場合はこれを使用します。

接続ピンの編集

IlvPinEditorInteractor クラスを使用すると、グラファー・ノードの接続ピンをインタラクティブに編集できます。このインタラクタがアクティブである場合、図 2.11 で示すように、ノードを選択するとその接続ピンが強調表示されます。

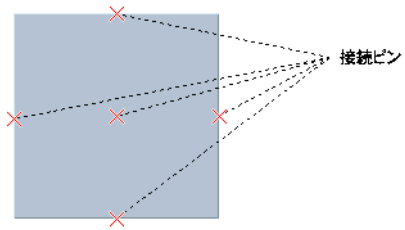

図 2.11 *強調表示された接続ピン*

グラファー・ノードを選択すると、次が可能になります。

- ノードの内側をクリックして新しい接続ピンを追加する。
- 接続ピンを削除する。これを行うには、マウスでピンを選択して Delete キーを押します。
- 既存の接続ピンを移動する。これを行うには、マウスでピンを選択して希望の位置までドラッグします。
- リンクをピンに接続、またピンから接続解除する。これを行うには、接続ピンを選択してから該当するリンクをクリックします。

> ***メモ:*** *作業中のノードが既にピン管理オブジェクトに関連付けられている場合、このオブジェクトは* `IlvGenericPin` *タイプでなければなりません。ノードが接続ピンを定義しない場合は、* `IlvGenericPin` *インスタンスが自動的に作成されます。*

リンクの編集

リンクを選択すると、その選択オブジェクトによりハンドルが描画されます。これを使用して形状を変えたり、接続方法を編集したりできます。図 2.12 は選択されているノードを示します。

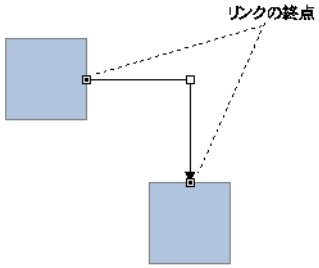

図2.12 *選択されたリンク*

終点ハンドルをドラッグして次を実行できます。

- リンクを付加する先の接続ピンを変更する。ハンドルを接続ピンの近くにドラッグすると、ピンは強調表示され、リンクはこの位置を利用して終点を計算します。
- リンクを他のノードに接続する。

中間点ハンドルを使用してリンクの形状を編集できます。これらのハンドルで許可されるインタラクションの種類は、編集中のリンクの種類によって異なります。

> ***メモ:*** *リンク編集は* `IlvGrapher::setLinksEditable` *メソッドを使用してオフに切り替えることができます。* `IlvGrapher` *インスタンスが作成されると、デフォルトではリンク編集が使用不可になります。*

索引

数字

A

C

D

G

I

L

M

N

O

P

め

り