DIERK KÖNIG

Dierk König（@mittie）：バーゼル（スイス）にあるCanoo Engineering AGのフェロー

OpenDolphin で実現するエンタープライズ JavaFX

エンタープライズ Java とデスクトップ Java の架け橋となる OpenDolphin

JavaFX テクノロジーは驚くほど魅力的なユーザー・インタフェースを実現します。しかし、JavaFX ファンは、次のエンタープライズ・アプリケーションをどのように開発すべきでしょうか。あるいは、既存のエンタープライズ・アプリケーションをどのように移行すべきでしょうか。OpenDolphin は、ビジネス・ロジックを視覚化機能から厳密に分離する無料のオープンソース・ライブラリです。ビジネス・ロジックと視覚化機能を分離することで、複数の異なる UI ツールキットの切り替えが容易になるだけでなく、これまでにビジネス・ロジックに対して行った投資がすべて保護されます。

JavaOne 2012 の Java 戦略基調講演において、OpenDolphin は JavaFX とエンタープライズ・アプリケーションの統合における成功事例として紹介されました。基調講演では、コンテナ・ヤードの監視、コンテナ配置変更計画の支援、およびその結果生成される作業計画の実施シミュレーションを行う対話型 3D アプリケーションについて、デモが実施されました（**デモ 1**）。

このコンテナ監視のデモは、コンテナ・ターミナル・ソフトウェアの世界最大手である Navis（CargoTec）が作成したもので、多数の JavaFX エンタープライズ・アプリケーションが持つ 2 つの重要な特徴について明らかにしています。

- 1 つ目に、開発者は JavaFX のリッチな UI 機能を十分に活用したいと考えている点、
- 2 つ目に、多くの場合サーバー中心型のプログラミング・モデルが長年利用されてきており、そのアーキテクチャを変えることはできないという点です。

OpenDolphin の概要

すべてのビジネス・アプリケーションがデータを扱うために最新かつ最高の 3D 機能を必要としているわけではありません（もっとも、多くのユースケースで 3D 機能は意外にも効果的ですが）。しかし、関心を引くアプリケーションを作成するためには、少なくともスマートフォンやタブレットで使い慣れたユーザー・エクスペリエンスには対応する必要があります。たとえば、状況依存型の入力補助、データの直接操作、滑らかな画面遷移、結果の瞬時の視覚化などが挙げられます。慣れたユーザー・エクスペリエンスよりも劣っていれば、魅力的には思えません。

デモ 2 に、金融ポートフォリオを管理するための従来型に近いアプリケーションのデモを示します。このデモは OpenDolphin に多数含まれているデモの 1 つです。デモ 2 のビデオを観れば、一見すると従来型のようなビジネス・アプリケーションでも、以下のような機能があれば魅力的なものになることが分かります。

- 結果の値を視覚化する全コントロールを、一貫性を保ちながら瞬時に更新
- フォームやダイアログ・ボックスなどの UI 要素の滑らかな表示 / 非表示（急な表示の切り替えはなし）
- アニメーション付きの画面遷移と、必要になったときに「具現化」されるコントロール
- 時間がかかっていないように感じるサーバー・ラウンドトリップ

このポートフォリオ・デモにおける魅力的なユーザー・エクス

デモ 1： JavaFX を使用した 3D コンテナ・ヤードに関するデモ

ペリエンスは、JavaFX と OpenDolphin を一緒に利用することで実現しています。OpenDolphin はプレゼンテーション・モデルの役割を担い、サーバーからの瞬時の非同期更新データを使用して、何を表示すべきかを決定します。アプリケーション・プログラマがこの OpenDolphin のプレゼンテーション・モデルに対して、アプリケーションの JavaFX ビューをバインドします。JavaFX ビューは、状態と画面遷移をどのように表示するかを決定する役割を担います。

表示される情報の一貫性を確保し、素早い更新を行うことが、OpenDolphin の機能です。さらに、視覚化用のコードの一貫性も高まり、かつ結合度は低くなります。ビューが他のビューを認識する必要はありません。また、システム内のすべてのビューに依存する「プレゼンタ」オブジェクトも存在しません。各ビューがそれぞれ 1 つのプレゼンテーション・モデルにバインドされます。OpenDolphin の概要は以上です。

移行リスクの軽減

OpenDolphin では、さまざまな種類の UI ツールキットが柔軟にサポートされます。JavaFX が推奨されますが、Swing や AWT、SWT、さらにクライアント側のフレームワーク（Eclipse RCP や NetBeans など）を使用することもできます。JavaBeans のような方法でリスナーとして登録できる Java ベースの UI ツールキットは、すべて問題なく動作します。

UI ツールキットの移行方法として、まず、OpenDolphin の支援を受けてビュー・レイヤーをカプセル化し、その後にそのビュー・レイヤーを置き換えます。今後 JavaFX に移行するための準備を行う場合でも、即座に JavaFX に移行しつつリスクを分散したい場合でも、この移行方法を使用することで作業を安全に進めることができます。

OpenDolphin のデモ・セクションには、パフォーマンス・デモと呼ばれるものが含まれます（**デモ 3**）。移行のしやすさを示すため、このデモには Swing バージョンと JavaFX バージョンが用意されています。ビジネス・ロジックの変更はまったく必要ありません。いずれのアプリケーションでもサーバー上では同じアプリケーション・コードが稼働し、ビューのみが異なります。

移行方法

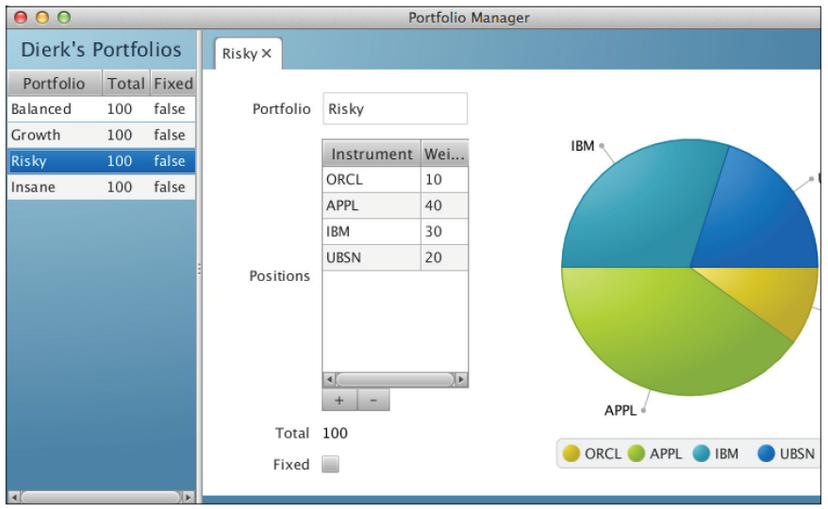
UI ツールキットの移行を実行するためには、まず、OpenDolphin の支援を受けてビュー・レイヤーをカプセル化し、その後にそのビュー・レイヤーを置き換えます。

変化への対応

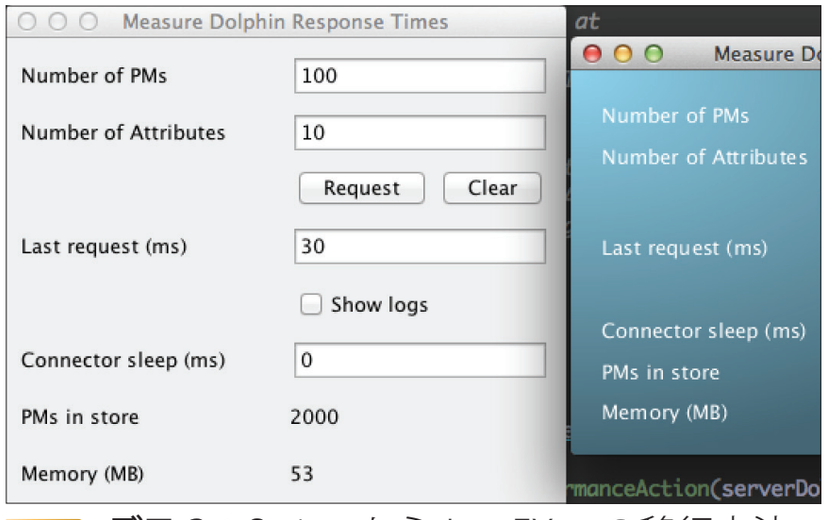
ビューを分離することで、ビューの作成、テスト、デバッグ、メンテナンスがはるかに容易になります。さらに重要なことに、変化する要件に対応するための適応と拡張も容易です。技術的な機能拡張やバージョンのアップグレード、新しいツールキットの導入などにより、変更が必要になることもあります。

あるいは、単純に新しいオプションを利用してみたい場合もあるでしょう。**デモ 1** のコンテナ・ヤードのデモを例にとると、このデモは当初、完全な 2D で開発されましたが、JavaFX の 3D 機能が登場し、アプリケーション・チームは 3D 機能の利用を要望しました。この要望に、ビュー・レイヤーの変更のみで対応でき、実行時の 2D モードと 3D モードの切り替えもサポートできました。

通常、移行の際には、デモの外観ばかりでなく、同時実行性の観点から見たツールキットの動作についても心配する必要があります。この同時実行性に関しても、OpenDolphin は優れています。視覚コンポーネントへのアクセスはすべて、UI スレッド内での実行が保証されます。また、コントローラのアクションはすべて、UI スレッド外での実行が保証されます。UI スレッドがブロックされることはありません。



デモ 2：ポートフォリオ管理デモ



デモ 3：Swing から JavaFX への移行方法を示すパフォーマンス・デモ

OpenDolphin ではビュー・オブジェクトとコントローラ・オブジェクトが厳密に分離され、それらのオブジェクト内で参照が共有されないため、誤ったスレッド処理が行われることは実質的にありません。ビュー・オブジェクトとコントローラ・オブジェクトは、同じクラスパス上にすらありません。

論理的分離と物理的分離

これまで、1 台のデスクトップ・マシン上での動作についておもに説明しましたが、エンタープライズ・アプリケーションはサーバー上で稼働します。そのため、OpenDolphin ではすべての制御ロジックを物理的なサーバーにインストールし、クライアント側のプレゼンテーション・モデルとサーバー側のプレゼンテーション・モデルを自動的に同期することが可能です。この分散アーキテクチャの概略図を**図 1** に示します。

OpenDolphin におけるモデル・ビュー・コントローラ（MVC）の分離は論理的な分離です。つまり、M（モデル）はドメイン・モデルではなくプレゼンテーション・モデルを表します。ビューとコントローラは常に厳密に分離され、異なるスレッドで実行され、コンパイルも別々に行われます。通常、ビューとコントローラの配置も異なるマシンに分散されます（クライアント側にビューが、サーバー側にコントローラが配置されます）。ただし、開発、テスト、デバッグの目的で、同じ Java 仮想マシン（JVM）内に配置することもできます。

ビューをクライアント側で実行することで、JavaFX の忠実性を十分に活用できます。サーバー側のコントローラでは、ビジネス・ロジックをエンタープライズ・コンテキスト内に配置します。このエンタープライズ・コンテキストには、ビジネス・ロジックが一元的に保管、管理されており、ビジネス・ロジックは企業のデータソースに最速でアクセスできます。

また、サーバーは、複数のクライアントを同期する際のハブの役割も果たします。列車制御デモ（**デモ 4**）および多数のイベントに関するデモ（**デモ 5**）ではこのハブ機能を多用しています。

これらのデモでも、サーバーのラウンドトリップに時間がかからないという印象を受けます。ポイントは、サーバーを非同期で呼び出し、ほんの少しずつデータを送信していることです。一般的なラウンドトリップ時間は 50 ミリ秒未満となります。さらに、非同期であることから、ユーザーの待ち時間は発生しません。

クライアント / サーバー間で交換される情報は、限定的なコマンドのセットだけです。そのようなコマンド・パターンを使用することで、サーバーが一時的に利用不可になったときのコマンドの一括実行、最適化の適用、再現、元に戻す / やり直し操作のサポートまでも可能になります。

さらに、実際の転送メカニズムや転送プロトコル、転送形式はプラガブルです。現在サポートされている転送メカニズムは HTTP REST およびインメモリです。

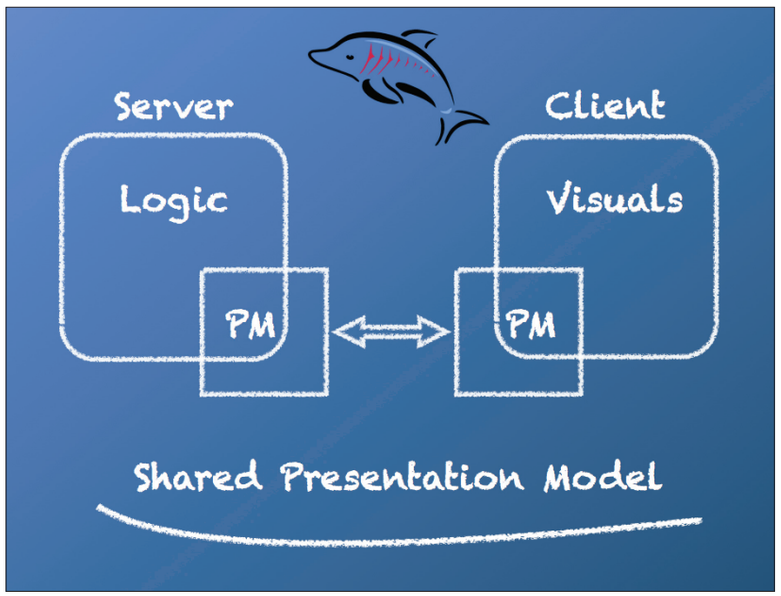
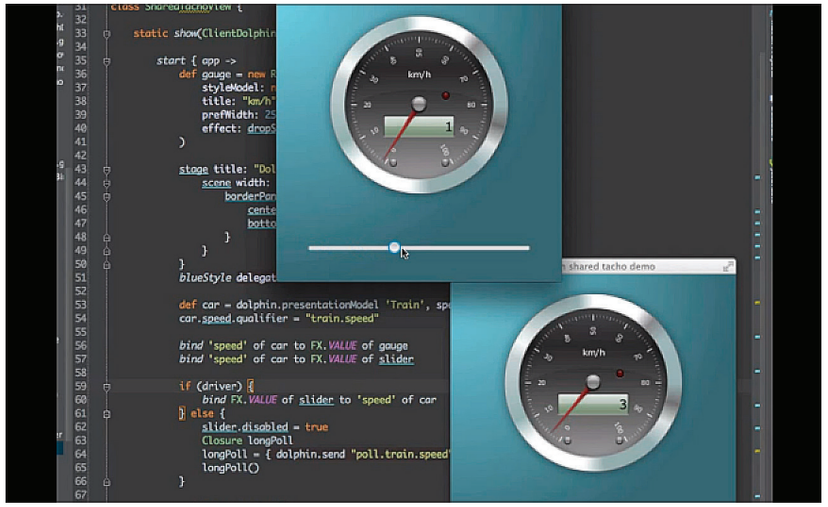


図 1



デモ 4：列車制御デモ

デモ 5：多数のイベントのデモ

まずは試すことから

OpenDolphin にはすべての機能のサンプルとして使用できる数多くのデモが付属しています。これらのデモは、手始めにシンプルな例から試してみる場合に便利です。その目的で、筆者らは GitHub 上に DolphinJumpStart プロジェクトを構築しています。このプロジェクトは、Maven または Gradle によるビルドの標準的なプロジェクト構造を提供するもので、プレーンな Java で記述された JavaFX ビューが含まれています（他のプロジェクトには、プレーンな Java ではなく GroovyFX を利用して手順を簡略化しているものもあります）。DolphinJumpStart プロジェクトにより、シンプルな JavaFX ビューから、クライアント / サーバーを分散させ十分に「Dolphin を駆使した」ビューまで、7 つのステップから成るシリーズを確認できます。

このプロジェクトについて簡単に説明します。以下のような JavaFX テキスト・フィールドを含むビューがあるとします。

```
TextField field =
  new TextField();
```

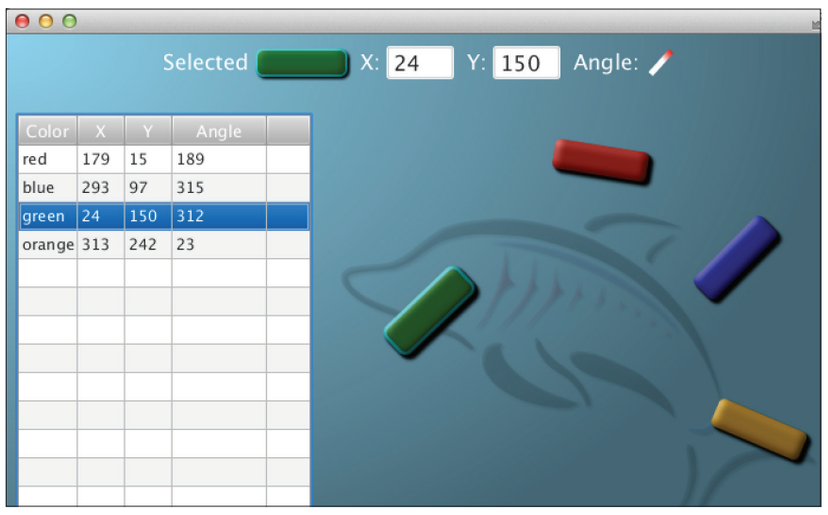
このフィールドの内容を OpenDolphin のプレゼンテーション・モデルにバインドします。そのためには、まずモデルが必要です。input という名前を使用してモデルを取得し、さらに text 属性を作成します（**リスト 1**）。

プレゼンテーション・モデルを作成した後、そのプレゼンテーション・モデルをビューにバインドできます（**リスト 2**）。バインディングには、テキスト・フィールドの内容が変更されるたびに、プレゼンテーション・モデルが更新され、サーバーに対して非同期に通知されるという効果があります。

このテキストの内容に対してサーバー側で何らかの操作（データベースへの保存など）を行うとします。説明を単純にするために、ここではテキストの内容を出力して、サーバー側で利用できる状態であることを確認します。サーバー側のアクションを**リスト 3** に示します。

このサーバー側のアクションは、次のコマンドを送信することでトリガーできます。

```
clientDolphin.send("PrintText");
```



デモ 6：プッシュ・デモ

リスト1　リスト2　／リスト3

```
PresentationModel input = clientDolphin.presentationModel(
"input", new ClientAttribute("text"));
```

すべてのリストのテキストをダウンロード

バインディングの力

前項までに、バインディングで実現できることを少しだけ学習しました。OpenDolphin にはもっと多くの魅力的なバインディング機能があります。たとえば、enabled、visible、mandatory、label、tooltip などの属性の dirty 状態に対してバインドできます。また、正規表現を使った属性固有のバリデーションも、クライアント側で実行できます。JavaFX で非常に詳細なプロパティ・バインディングを行い、OpenDolphin で情報ソースを指定するという方法も有効です。

しかし、OpenDolphin には、安定的なバインディングという非常に優れた機能があります。ビジネス・アプリケーションは、マスター / ディテール関係を持つビューにあふれています。マスター / ディテール関係が明白な場合もありますが、タブ・ペインや内部フレーム、ナビゲーション・バーなどで分かりにくくなっている場合もあります。プッシュ・デモ（**デモ 6**）はその良い例です。課題は、すべてのビューが他のビューについて認識せず、すべてのビューを制御する中央の「マネージャ」も存在せず、過度のアンバインド処理やリバインド処理を実行しなくても、一貫性のある更新を正しく実行できるようにすることです。OpenDolphin を使用した場合、こうした問題は、発生することすらありません。すべてのビューは専用のプレゼンテーション・モデルをただ 1 つ認識し、このバインディングが変更されることはありません。

オープンなプロジェクトにご参加を

OpenDolphin プロジェクトにはどなたでもご参加いただけます。概要については、ユーザー・ガイドのアーキテクチャ・セクションをお読みください。1.0

のリリース後も、OpenDolphin の API では後方互換性が維持されます。

ビデオ、スクリーンキャスト、技術プレゼンテーションなどのさまざまな OpenDolphin のリソースが Web で公開されています。OpenDolphin は Apache 2 ライセンスの下で管理されるオープンソースであり、GitHub 上でホストされています。ご質問、アドバイス、その他のご協力をお待ちしています。

OpenDolphin を使用して構築された初期の大規模アプリケーションは現在本番運用の段階にあり、また、本番運用間近のアプリケーションもあります。早期に導入された皆様にはこれまで多くの刺激をいただき、このライブラリの機能やパフォーマンスの改善に貢献していただきました。また、早期導入時に発見されたこととして、OpenDolphin のアプローチがテストの自動化に非常に有用であるという点があります。UI モックや機能モックを無料で手に入れることができるからです。

また、機能テストの目的では、クライアントのプレゼンテーション・モデルを作成し、コマンドを送信し、モデル・ストア内での結果を検証するだけでテストできます。ある開発者の言葉を借りれば、「これは UI を使用しない UI テスト」です。

作業が容易に

ビューの作成、テスト、デバッグ、メンテナンスがはるかに容易になります。さらに重要なことに、変化する要件に対応するための適応と拡張も容易です。

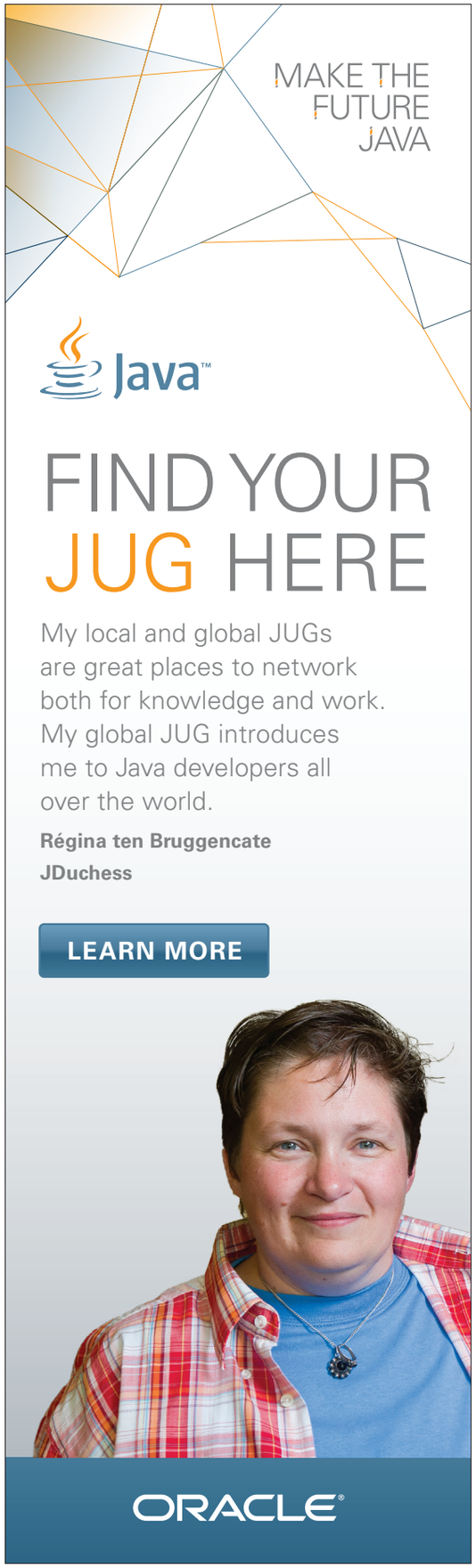
まとめ

UI の構成作業はしばしばソフトウェア・エンジニアリングの厄介な作業のように見られ、多くのアプリケーションでは漠然とモデル / ビュー / コントローラに言及する以外に、プレゼンテーション・コードの構造に対してはほとんど労力が費やされていません。OpenDolphin では、汎用的なプレゼンテーション・モデルを信頼のおける唯一の情報源として使用することで、ビューを厳密に分離し、コントローラを完全に切り離すことができます。すべての通信が非同期に実行され、また、自動的に適切なスレッド内で実行されます。

エンタープライズ・アプリケーションと同様に、OpenDolphin もサーバー上で稼働します。そのため、データのセキュアな共有、稼働中の更新、マルチユーザー対応、共同作業のチャンスが無限に得られます。これらの機能を世界有数の視覚化テクノロジーと組み合わせるために、OpenDolphin があります。

JavaFX の世界に乗り込もうとしている場合や、変化する UI テクノロジーからビジネス・ロジックへのこれまでの投資を保護したい場合、あるいはビジネス・アプリケーションのエンタープライズ規模での準備を模索している場合にも、OpenDolphin の利用を真剣に検討することをお勧めします。

OpenDolphin は、Raspberry Pi のようなデバイスに組み込んでも実行できるほど小さく軽量です。筆者が自信を持って言えることは、OpenDolphin が有望な技術として Web の世界に登場したときのように、JavaFX mobile を使用した未来のモバイル・コンピューティングにも OpenDolphin が対応できるということです。**</article>**

LEARN MORE

- Twitter @OpenDolphin
- Dolphin 参考ドキュメント
- git@github.com:canoo/open-dolphin.git
- git@github.com:canoo/DolphinJumpStart.git
- Dierk König の YouTube ビデオ