



OpenMeisterEnterprise® .NET 2015 製品説明書

2020年10月05日

日本電気株式会社

1. OpenMeisterEnterprise .NET 2015とは？
2. OpenMeisterEnterprise .NET 2015の目的
3. 開発ガイド
4. システム基盤
5. 導入効果
6. 動作確認環境
7. 製品サポート／保守サービス

【参考資料】

OpenMeisterEnterprise .NET 2015のアーキテクチャ ... etc

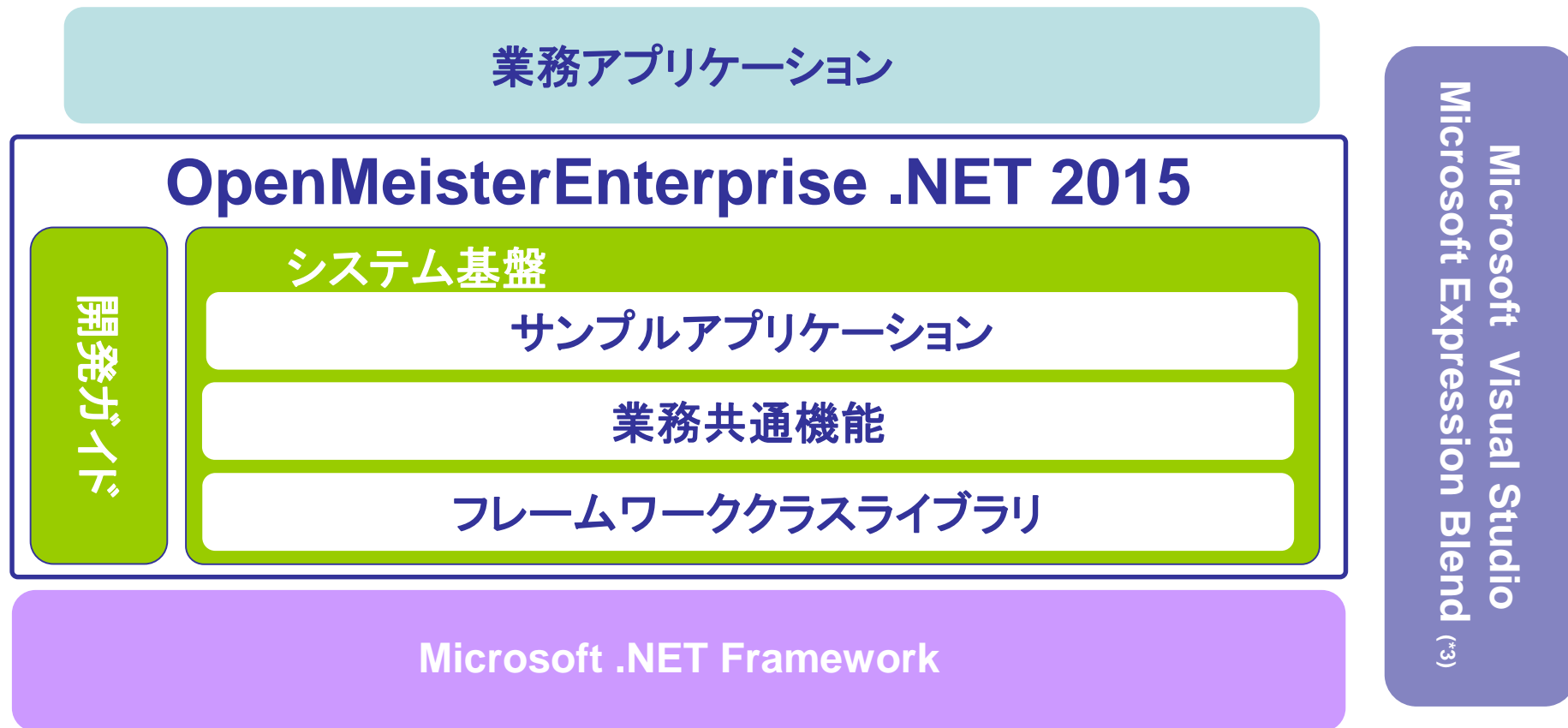
※商標について

本書に記載している会社名・製品名などは、各社の商標または登録商標です。

(本文中には™や®は記載していません)

1. OpenMeisterEnterprise .NET 2015とは？

.NET Framework^(*1) 向けの
業務アプリケーションを開発するための フレームワーク。
「システム基盤」と「開発ガイド^(*2)」があります。



※対象バージョンの詳細に関しては、「6. 動作確認環境」を参照してください。

(*1) .NET Framework 1.x は対象外です。 (*2) .開発ガイドの製品名はOpenMeisterEnterprise.NET 2005 です。

(*3) Expression Blend は、WPF での ユーザインターフェース作成時のみ。

2. OpenMeisterEnterprise .NET 2015の目的

業務アプリケーション開発を成功させるためには、検討／準備すべきことがたくさんある！！



品質の良い業務アプリケーション開発に必要な「開発標準」「システム基盤」を提供！！

①一定の品質を確保

標準的な設計／実装手段を規定し、設計／実装手段が担当者に依存しないようにする。

②業務アプリケーションライフサイクル全体の生産性向上／技術的リスク低減

通信処理, トランザクション処理, 例外処理etc 必要な非機能要求処理は、実装／評価済み。

業務ロジック(機能要求)部分の開発に集中できる。

効果の高い機能に厳選した **軽量フレームワーク** で、業務アプリケーション開発者から直接見える機能、適用に必要なスキル(覚えなければならないこと)を極力減らしています。

3. 開発ガイド

開発手順や成果物の標準化のための

「テンプレート」や「ガイドライン」を提供しています。

【ドキュメントテンプレート・サンプル】

<要件定義フェーズ>

対象領域図

<システム分析／標準化フェーズ>

E-R図

コード設計書

データ項目一覧

機能一覧

画面設計書

論理スキーマ設計書

<システム設計フェーズ>

テーブル設計書

ビュー設計書

外部設計書

画面設計書

<製造／単体テストフェーズ>

製造結果報告書

<結合テストフェーズ>

結合テスト仕様書兼報告書

<総合テストフェーズ>

総合テスト仕様書兼報告書

<保守／運用フェーズ>

基本バックアップ運用説明書

基本障害運用説明書

セキュリティ仕様

<その他>

標準文書テンプレート

標準文書テンプレート使用説明書

【規約・規則】

<C#、Visual Basic>

命名規則 (C#、Visual Basic)

コーディング規約 (C#、Visual Basic)

コメントコーディング規約 (C#、Visual Basic)

<データベース>

命名規則 (データベース)

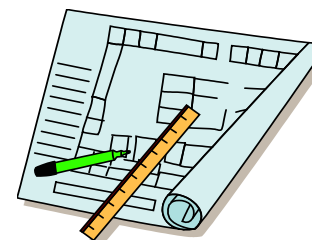
コーディング規約 (SQL)

<その他>

システム規約

命名規則 (共通)

ローマ字綴り方規約



【標準・ガイドライン】

<画面>

画面設計標準

画面設計標準説明書

画面遷移標準

<データベース>

DB設計標準

DBバックアップガイドライン

DBセキュリティガイドライン

<運用>

ロギングガイドライン

トレースガイドライン

<その他>

システム開発手順書

メッセージ標準

性能設計ガイドライン

セキュリティガイドライン



4. システム基盤

業務アプリケーション開発での「技術的リスク低減」「品質／生産性向上」を目的にシステム基盤を提供しています。

フレームワーククラスライブラリ（dll形式）

技術的に難易度の高い機能をあらかじめご用意
（軽量で分かりやすく、適用に工数がかからない！）

システムを開発する上で最低限必要な基本機能をもつ、クラスライブラリ。

業務共通機能（ソースコード形式）

プロジェクトの非機能要件に応じた「アーキテクチャ設計」や「開発基盤の整備」の、技術的リスクと作業工数を低減！

業務アプリケーションで共通に利用する機能を集めたクラスライブラリ。

アーキテクチャも規定します。

ソースコード（C#, VB.NET）形式ですので、業務の特性に応じて、追加や改造も可能です。

サンプルアプリケーション（ソースコード形式）

4階層の設計／実装により、変化に強いアプリケーション開発が可能！

「フレームワーククラスライブラリ」「業務共通機能」を利用した、サンプルアプリケーション。

OME.NETの基本な処理の流れと、ライブラリの具体的な利用例をすぐにご確認いただけます。

開発要員のスキルアップ教材としてもご利用ください。

4. システム基盤 ～フレームワーククラスライブラリ～

システムを開発する上で、最低限必要な基本機能を
フレームワーククラスライブラリとして提供しています。

主な機能	説明
サービス呼出機能 (Service)	サーバロジック呼出 .NET RemotingやWebサービス、またはその他通信方式といったさまざまな実装で公開されるサービスを、呼出側から詳細を意識せずに実行するための機能など 呼び出し実行時に、処理時間の計測や「お待ち下さい」ダイアログの表示を自動的に行う機能など
ファサード実装用 基本機能 (Interception)	設定情報適用 構成ファイルで指定された設定情報を自動的にファサードクラスに適用する機能など ファサード呼出時の割込制御 ファサードで定義したメソッドが呼び出される際に自動的に割り込み処理を行う機能など 実行時間計測 ファサードで定義したメソッドが実行される際に、メソッドの処理にかかった時間を自動計測する機能など
設定関連機能 (Configuration)	構成ファイルによるプロパティ設定 特定のインスタンスに対するプロパティの設定機能、独自クラスに対する構成ファイルによるプロパティの設定機能など
メッセージ取得機能 (Message)	メッセージIDからメッセージ文字列を取得 リソースや構成ファイルなど実際のメッセージの格納先を意識せずに、IDで指定されたメッセージの取得を行う機能など メッセージのプレースホルダにパラメータを埋め込む機能、メッセージを自由に整形するためのカスタマイズ機能など メッセージ取得先として、テキストファイル、CSVファイル、XMLファイル、リソース、構成ファイルなどに対応する機能など
ユーティリティ機能 (Utility)	タイムスタンプ 特定時点のタイムスタンプ取得、メソッドなど特定区間での実行時間やガベージコレクション回数の計測など、簡易なパフォーマンスの計測と出力を行う機能など
自動ダイアログ表示機能 (Windows)	「お待ちください」ダイアログの自動表示 ある程度以上の時間がかかることが予想される処理を実行する場合に、必要に応じて「しばらくお待ちください」といったダイアログを自動表示する機能など

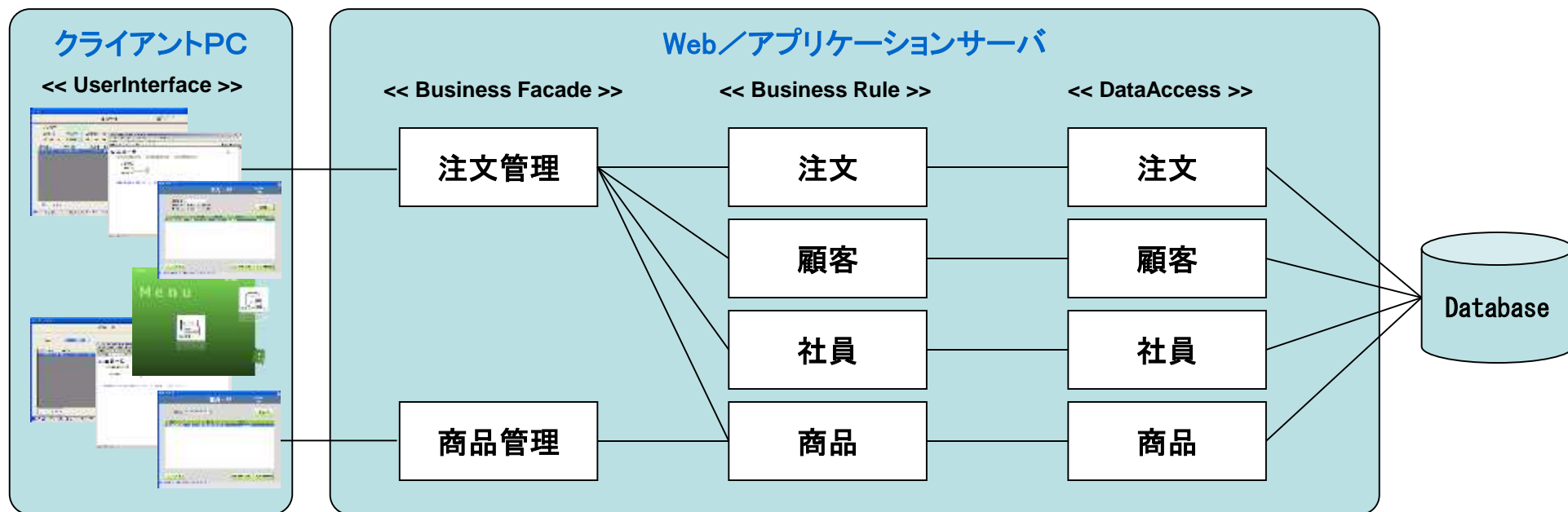
4. システム基盤 ～業務共通機能～

アプリケーションのアーキテクチャおよび各業務で共通に利用する機能を提供しています。
ソース(C#,VB.NET)提供ですので、業務の特性に応じて、追加や改造も可能です。

主な機能	説明
ファサード呼出機能 (Service)	<u>.NET Remotingを利用したファサード呼出</u> .NET Remotingを利用して、サーバで公開されるファサードを呼び出す機能。通信エラー時の再試行、認証情報の自動送信、認証タイムアウト時に再ログインダイアログを表示する機能など
基底ファサード (Facade)	<u>業務共通の基底ファサード</u> ファサードを実装するための共通基底クラス。ファサードのメソッドが呼び出される際に、ユーザの認証、アクセス制御、トランザクションの管理、例外発生時のログ出力を行う機能など
共通情報管理 (Information)	<u>現在のユーザ情報等、業務に必要な各種情報の管理</u> アプリケーション共通の情報や現在のユーザ情報、サーバ・クライアントの共有情報など、共通的に使用される情報の管理を行う機能など
ログ出力機能 (Logging)	アプリケーションで検知した障害や動作の履歴など、任意の情報をテキストファイルに出力する機能、出力する情報の重要度によるログ出力の有効／無効の切り替えを行う機能、日付で出力先ファイルを切り替える機能など
認証関連機能 (Security)	ユーザのログイン、チケット認証、ユーザ情報のセットアップ機能など
ユーティリティ機能 (Utility)	例外オブジェクトに対する追加情報（例外追跡用ID、ユーザ表示用メッセージ、その他例外処理用の補足情報等々）を設定または取得する機能、Shift-JIS変換不可文字(JIS2004文字等)のチェック機能など
画面共通機能 (UserInterface)	業務共通の基底フォーム、ログイン用ダイアログ、例外処理をまとめて行う集約例外ハンドラの機能など ボタンなどのアプリケーション起動用のコントロールに、メニュー情報（起動アプリケーション情報）を登録する機能など 入力コントロール等に対して、入力検証用情報（検証ID等）の登録と、登録した情報にしたがって検証を実行する機能など
リモートイング関連機能 (Remoting)	.NET Remotingによる通信において、通信データの圧縮、通信帯域制限のエミュレーション、通信の簡易暗号化を行う機能など .NET Remotingで公開するサービスを一括で登録する機能など
入力値検証機能 (Validation)	検証IDを指定することで、IDに対応するルールにしたがって項目の入力値などを検証する機能

4. システム基盤 ～サンプルアプリケーション～

サンプルアプリケーションとして、注文システム(注文管理機能/商品管理機能)を提供。
ユーザインタフェースはWindowsForm、WebForm、WPFを提供(Facade以降のロジックは共通)。

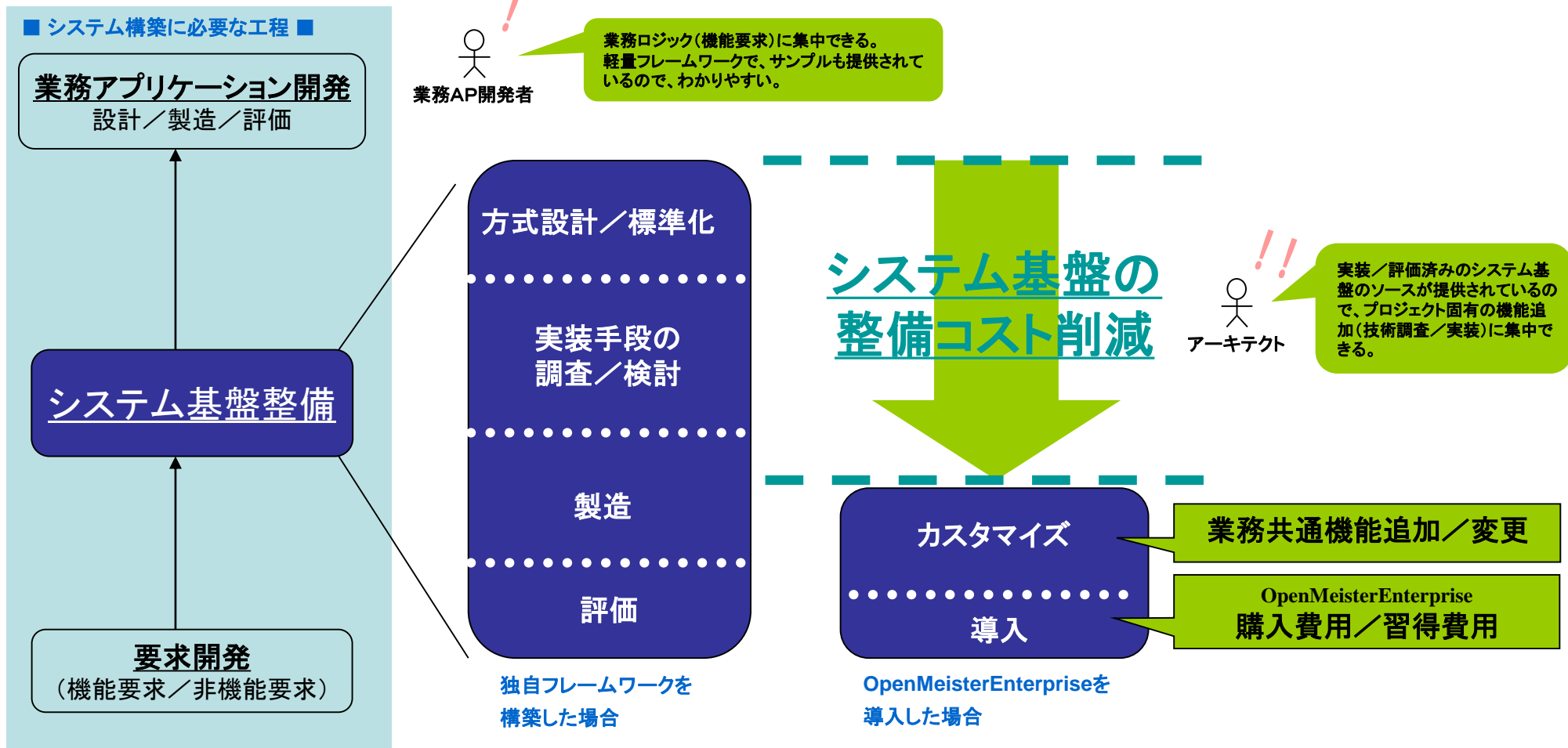


業務アプリケーションで行うべき処理を、論理的に大きく4階層に分けています。
各層それぞれの責務&インタフェースを明確にして、具体的な内部構造やロジックをカプセル化することで隠蔽し、内部の変更による影響を他層に与えにくくします。

- ①UserInterface層 ユーザとのデータ入出力手段である画面の制御を行います。
- ②Facade層 UserInterface層からの要求の入口となります。
必要なRuleを組み合わせて、サービスのワークフローの制御を行い、
まとまったトランザクション単位の業務処理を行います。
- ③Rule層 業務上重要なビジネスロジックの制御を行います。
- ④DataAccess層 データアクセスを行います。Rule層に対して、データの永続化手段を隠蔽します。

5. 導入効果

業務アプリケーション開発のための **システム基盤整備** の「コスト削減」「期間短縮」「技術的リスク低減」を実現します。



6. 動作確認環境

実行環境(サーバ)

実行環境	評価環境	
APサーバ	(for SQL Server)	(for Oracle)
	Microsoft Windows Server 2019 Microsoft Internet Information Services (IIS) 10.0 ASP.NET 3.5 (Microsoft .NET Framework 3.5.1) ASP.NET 4.7 (Microsoft .NET Framework 4.8)	Microsoft Windows Server 2019 Microsoft Internet Information Services (IIS) 10.0 ASP.NET 3.5 (Microsoft .NET Framework 3.5.1) ASP.NET 4.7 (Microsoft .NET Framework 4.8) Oracle Data Access Components(ODAC) 19c (19.3)
DBサーバ	Microsoft Windows Server 2019 Microsoft SQL Server 2019 Developer	Microsoft Windows Server 2019 Oracle Database 19c (19.3)

実行環境(クライアント)

実行環境	評価環境
クライアント PC	Microsoft Windows 10 Microsoft Edge / Microsoft Internet Explorer 11 Microsoft .NET Framework 3.5.1 / 4.8

(*) 上記の環境はOpenMeisterEnterprise.NET 2015 Ver. 2.1リリース時（2020年10月）の動作確認環境であり、今後追加する可能性があります。

6. 動作確認環境

開発環境

開発環境	評価環境
開発PC	Microsoft Windows 10 Microsoft Visual Studio Professional 2015 Update 3 + Microsoft .NET Framework 3.5.1 / 4.8 Microsoft SQL Server 2016 Express LocalDB Oracle Data Access Components(ODAC) 19c (19.3)

(*) 上記の環境はOpenMeisterEnterprise.NET 2015 Ver. 2.1リリース時（2020年10月）の動作確認環境であり、今後追加する可能性があります。

7. 製品サポート／保守サービス

製品サポート

<https://jpn.nec.com/ome/>

製品情報／サービス情報を提供しています。

製品に対するご質問やご意見は、上記サイト内のお問い合わせ窓口までご連絡ください。

保守サービス

製品購入時に保守契約いただきますと、OpenMeisterEnterprise .NET 2015 に関する技術的なお問合せやバージョンアップ等の保守サービスを受けられます。

▼ ライセンスサービス

製品の無償バージョンアップ

▼ レスポンスサービス

製品に関する技術的なお問い合わせ対応

▼ インフォメーションサービス

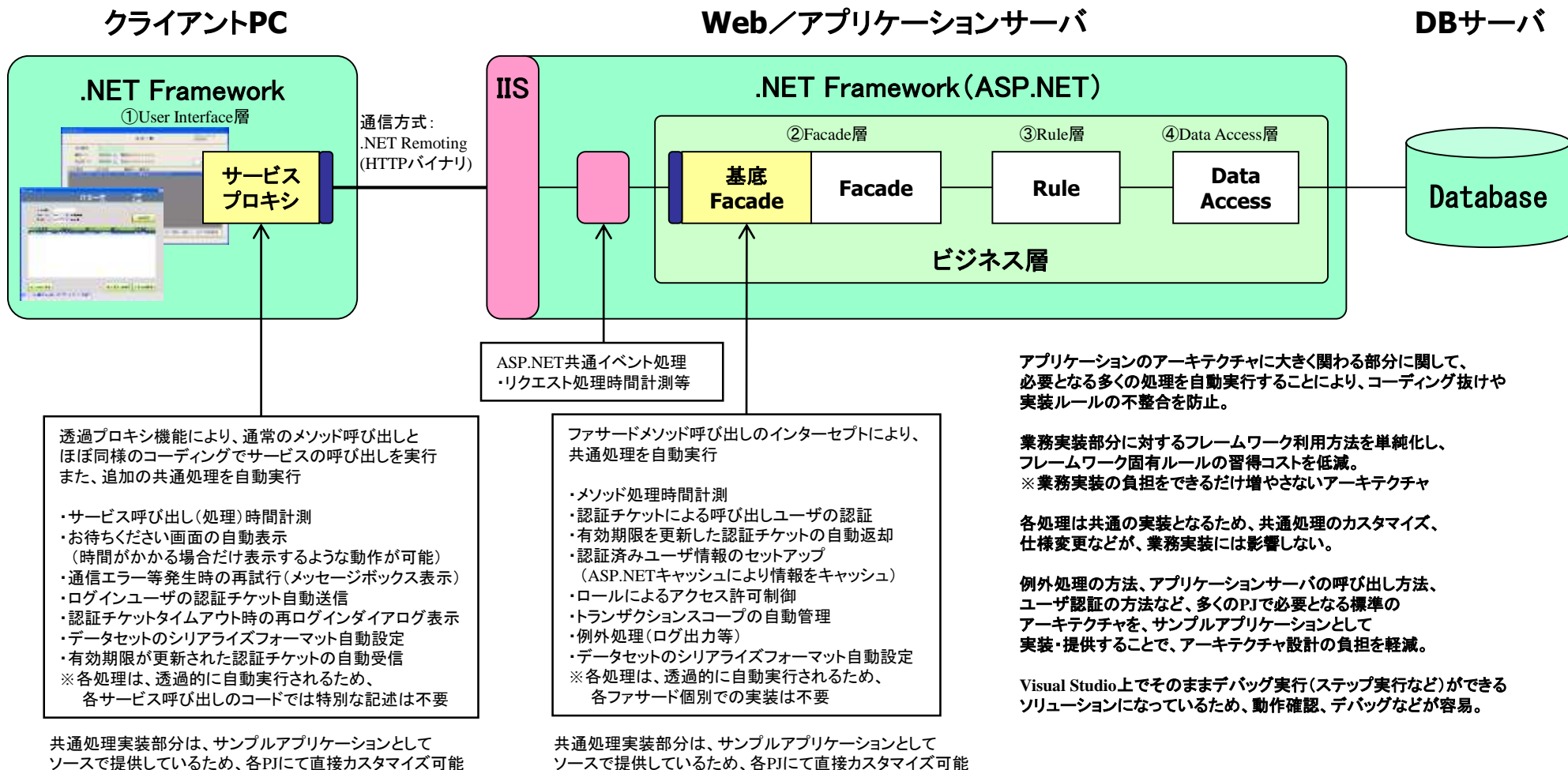
製品情報／技術情報の提供



参考資料

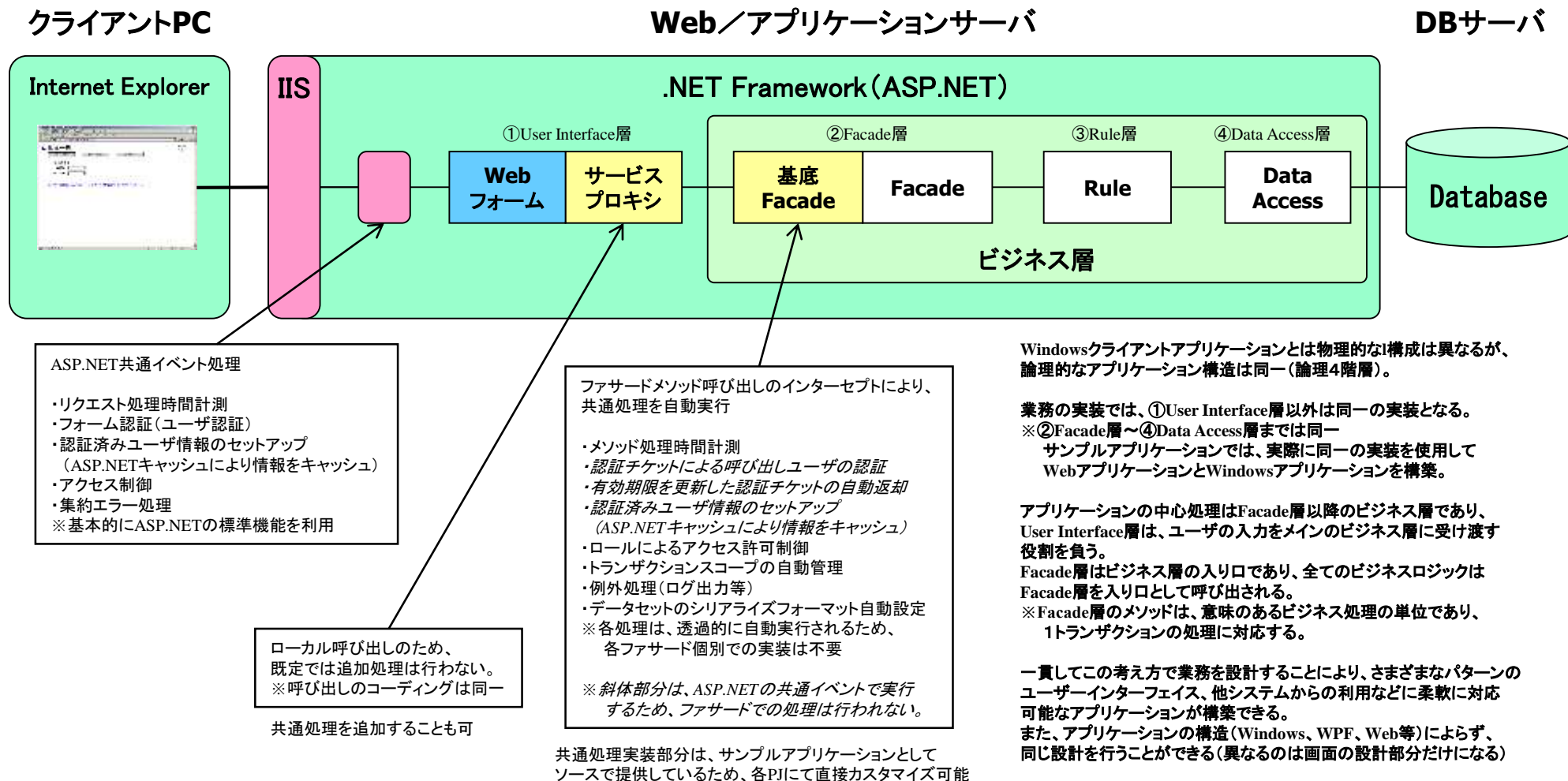
OpenMeisterEnterprise .NET 2015のアーキテクチャ

(1) Windows/WPFクライアントアプリケーション形式



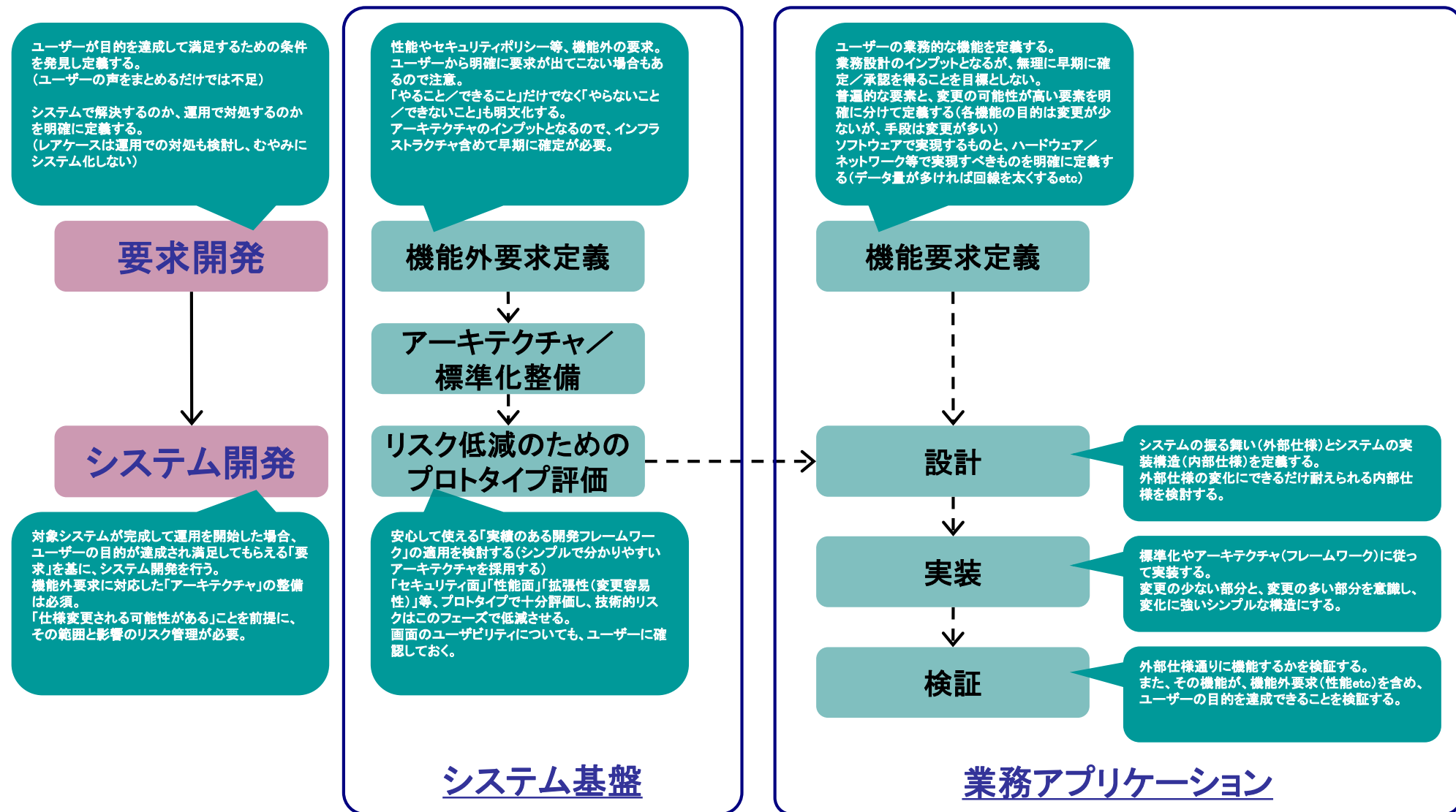
OpenMeisterEnterprise .NET 2015のアーキテクチャ (2)

(2) Webフォーム形式 (Webアプリケーション)

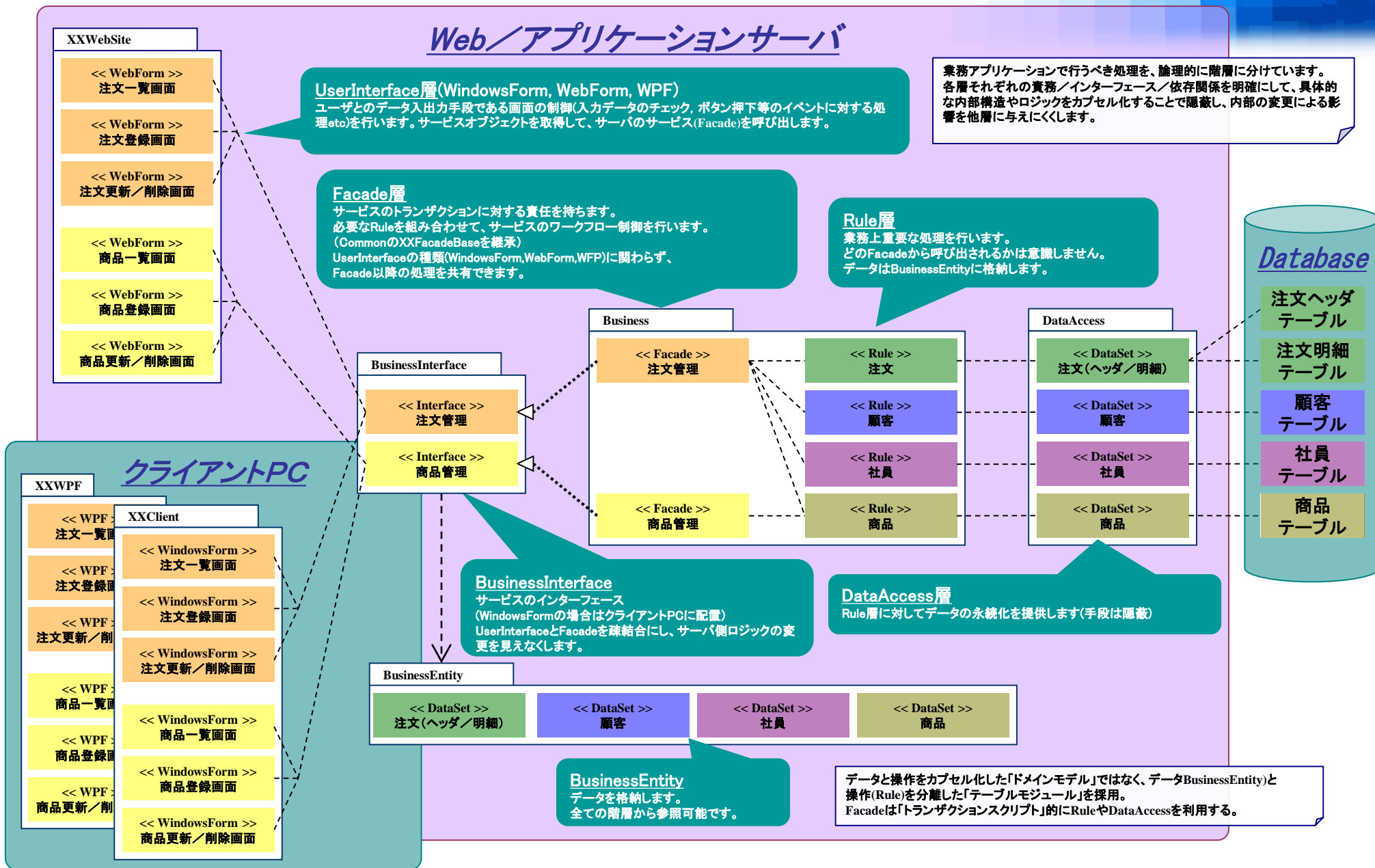


業務アプリケーション開発の流れ

業務アプリケーションの品質には、システム基盤の品質が大きな影響を与えます。



業務アプリケーション構成（各層の責務）



サンプルアプリケーションのコード例 (WindowsFormサンプルアプリケーションより抜粋)

ユーザと対話するために画面を制御し、必要に応じてFacadeを呼び出す。

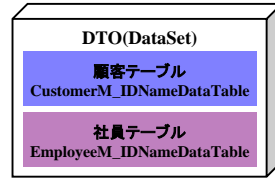
画面からの要求に対して責任を持ち、必要なRuleを適切な順序で呼び出す。

Facadeからの要求に対する業務ロジック処理。永続化はDataAccessに任せる。

```
<< UIInterface >> FormOrderList 注文一覧画面
private IOrderController m_orderController; //注文管理サービス
private DsCustomer.CustomerM_IDNameDataTable m_customerList; //顧客リスト
private DsEmployee.EmployeeM_IDNameDataTable m_receiverList; //担当者リスト
private DsOrder.OrderTDataTable m_orderList; //注文情報リスト
初期画面表示
FormOrderListのLoadイベントハンドラ(・・・)
{
    // 注文管理サービスオブジェクトを取得
    m_orderController = ServiceManager.GetService<IOrderController>();
    // 初期表示データを取得(ComboBox表示用)
    DataSet initData = m_orderController.GetInitialDataFormOrderList();
    DataUtility.GetTable(initData, "CustomerM_IDName", out m_customerList);
    DataUtility.GetTable(initData, "EmployeeM_IDName", out m_receiverList);
    // ComboBoxにセット
    ...
}
```

画面をWPFで構築した場合でも、JavaのWebサービス呼出でも同様

```
<< Facade >> OrderController 注文管理
初期表示データ取得
public DataSet GetInitialDataFormOrderList()
{
    // 顧客情報、社員情報をRuleから取得し、UIへ返却するコンテナに格納
    Rule.Customer c = new Customer();
    Rule.Employee e = new Employee();
    DataSet container = new DataSet();
    container.Tables.Add(c.GetCustomerIDNameList());
    container.Tables.Add(e.GetEmployeeIDNameList());
    return container;
}
```



```
<< Rule >> Customer 顧客
顧客リスト取得
public BusinessEntity.DsCustomer.CustomerM_IDNameDataTable GetCustomerIDNameList()
{
    DataAccess.DsCustomerMTableAdapters.CustomerM_IDNameTableAdapter ta
    = new DataAccess.DsCustomerMTableAdapters.CustomerM_IDNameTableAdapter();
    // 取得したデータを、返却用Entityにマージ
    BusinessEntity.DsCustomer.CustomerM_IDNameDataTable table
    = new BusinessEntity.DsCustomer.CustomerM_IDNameDataTable();
    table.Merge(ta.GetData());
    return table;
}
```

```
<< Rule >> Employee 社員
社員リスト取得
public DsEmployee.EmployeeM_IDNameDataTable GetEmployeeIDNameList()
{
    DataAccess.DsEmployeeMTableAdapters.EmployeeM_IDNameTableAdapter ta
    = new DataAccess.DsEmployeeMTableAdapters.EmployeeM_IDNameTableAdapter();
    // 取得したデータを、返却用Entityにマージ
    DsEmployee.EmployeeM_IDNameDataTable table
    = new DsEmployee.EmployeeM_IDNameDataTable();
    table.Merge(ta.GetData());
    return table;
}
```

```
<< Rule >> Order 注文
注文情報リスト取得
public DsOrder.OrderTDataTable GetOrderList(Hashtable argKeys)
{
    DataAccess.DsOrderTableAdapters.OrderTTableAdapter ta
    = new DataAccess.DsOrderTableAdapters.OrderTTableAdapter();
    DataAccess.DsOrder.OrderTDataTable daTable
    = new DataAccess.DsOrder.OrderTDataTable();
    DsOrder.OrderTDataTable table = new DsOrder.OrderTDataTable();
    // 検索キー指定(注文番号、顧客ID、担当者ID)
    string num = argKeys["OrderNumber"].ToString();
    string cstID = argKeys["CustomerID"].ToString();
    string recID = argKeys["ReceiverID"].ToString();
    if (num == "")
    {
        if (cstID == "" && recID == "") daTable = ta.GetData();
        if (cstID != "" && recID == "") daTable = ta.GetDataByCustomerID(cstID);
        if (cstID == "" && recID != "") daTable = ta.GetDataByReceiverID(recID);
        if (cstID != "" && recID != "") daTable = ta.GetDataByCustomerIDReceiverID(cstID, recID);
    }
    else {
        daTable = ta.GetDataByOrderNumber(num);
    }
    // 取得したデータを、返却用Entityにマージ
    table.Merge(daTable);
    return table;
}
```

業務アプリケーションではコーディングしていないけどどこでやってるの？

Facade呼び出しの通信や「お待ちください」画面表示は？
認証されていないクライアントからの要求拒否は？
ログインユーザ情報の受け渡しは？
トランザクション制御は？
例外Catch処理は？ ... 等々

→ 次ページ「システム基盤の役割」参照



```
検索ボタン押下
buttonRetrievalのClickイベントハンドラ(・・・)
{
    // 検索キーセット(注文番号、顧客ID、担当者ID)
    Hashtable keys = new Hashtable();
    keys.Add("OrderNumber", textBoxOrderNumber.Text);
    keys.Add("CustomerID", comboBoxCustomerID.Text);
    keys.Add("ReceiverID", comboBoxReceiverID.Text);
    // 注文情報リストを取得
    m_orderList = m_orderController.GetOrderList(keys);
    // 取得した注文情報リストを一覧表示
}

更新／削除ボタン押下
buttonGoOrderUpdateのClickイベントハンドラ(・・・)
{
    // 注文更新／削除画面表示
    using (FormOrderUpdate fm = new FormOrderUpdate(データ))
    {
        DialogResult result = fm.ShowDialog();
        if (result != DialogResult.Abort)
        {
            // 処理
        }
    }
}
```

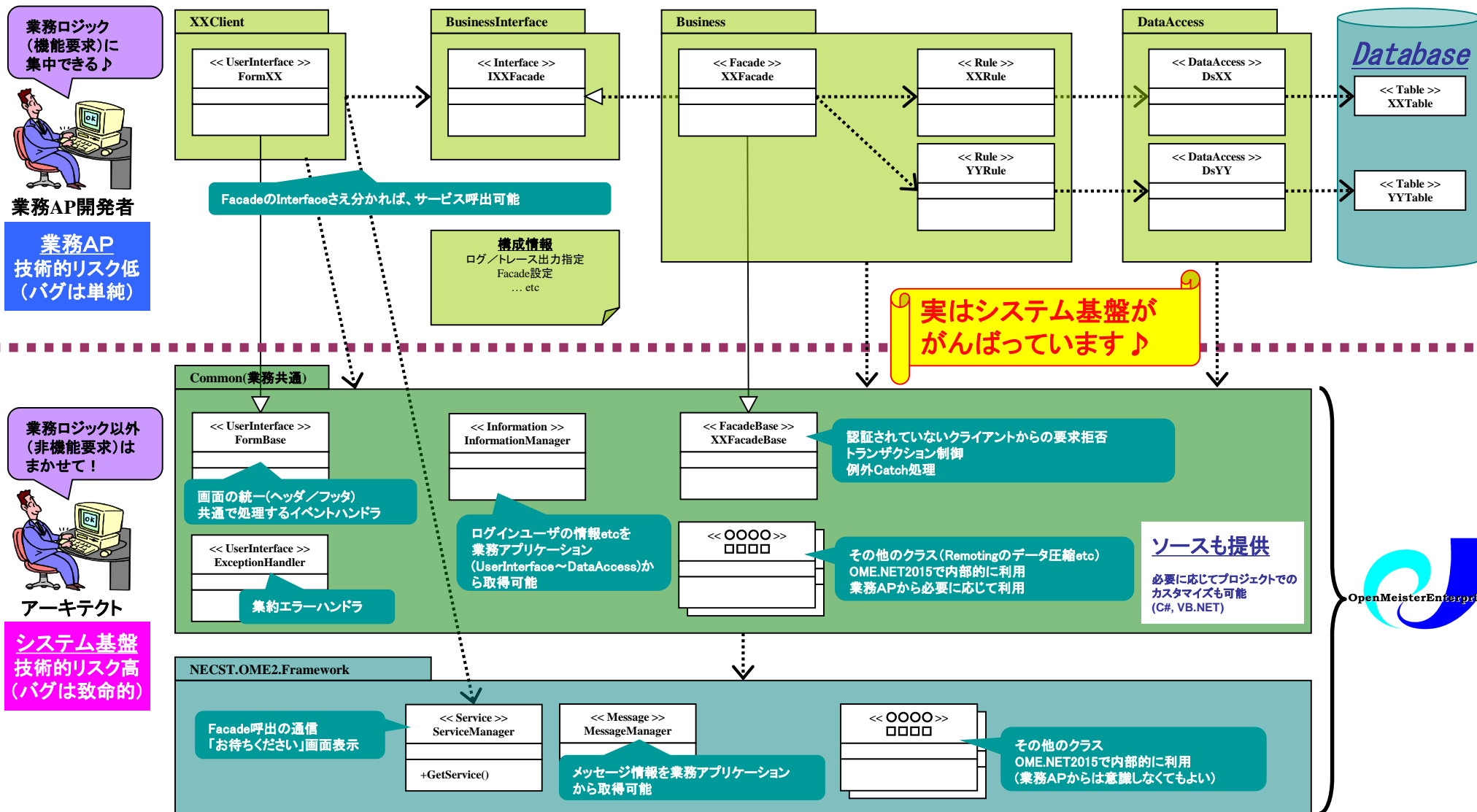
画面遷移やUIコントロールの制御は、一般的なコーディングで行う。

```
注文情報リスト取得
public DsOrder.OrderTDataTable GetOrderList(Hashtable argKeys)
{
    // 検索キーに合致する注文ヘッダリストを取得
    Business.Rule.Order o = new Order();
    return o.GetOrderList(argKeys);
}
```

注文ヘッダテーブル
OrderTDataTable

システム基盤の役割

業務APの関心事以外(通信手段etc非機能要求)は、できるだけシステム基盤で処理します。 → 技術的リスク低減, 品質/生産性向上
また、業務APから直接見えるシステム基盤はできるだけ少なくします。 → 導入コスト低減



各層間のデータ受け渡し

UserInterface～Facade

クライアントとサーバ間は通信が発生するので、できるだけ「呼び出し回数」を抑える必要がある。
→ DTO(DataSet等)に複数のEntity(DataTable等)を格納する(DTO内部の項目変更時にインターフェースの変更不要)。
(例) 初期画面表示で、コンボボックスに表示する「顧客」と「商品」のリストを一度に取得。

なぜ？

注文ヘッダDataAccessと注文明細DataAccessに分けない。
→テーブルをヘッダと明細に分けているのはリレーショナルDBに格納しているためであり、Ruleから意識したくないため。

Facade～Rule

Entityでのデータ受け渡しを基本とする。

Rule～DataAccess

DataAccessのDataSetでのデータ受け渡しを基本とする。
RuleでEntityに格納する。

なぜ？

注文登録に必要な商品のデータを取得する際に、商品Ruleを注文管理Facadeから呼び出している。
→注文画面からの処理は、全て注文管理Facadeが責任を持って処理するため。

なぜ？

DataAccessのDataSetをそのまま使わず、EntityのDataSetに格納。
→DataAccessにはSQLを含んだTableAdapterクラスが含まれており、それを他層に渡したくない。

Database

注文ヘッダ
テーブル
注文明細
テーブル

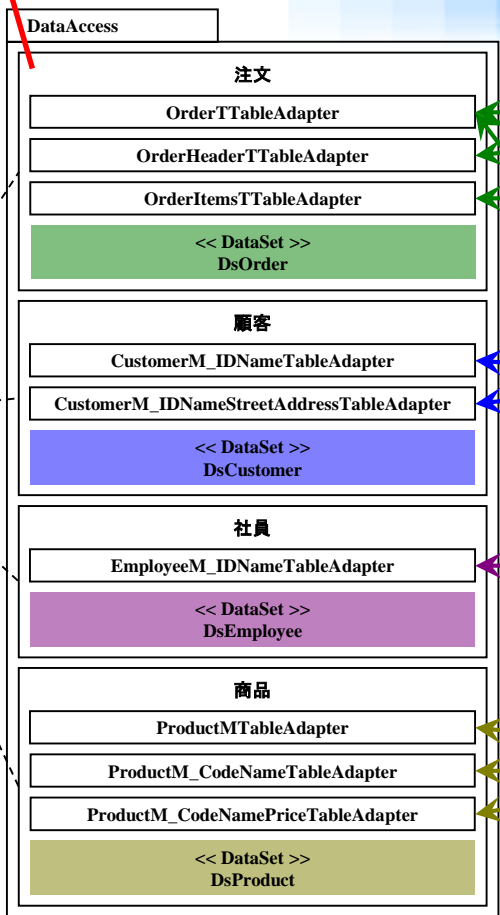
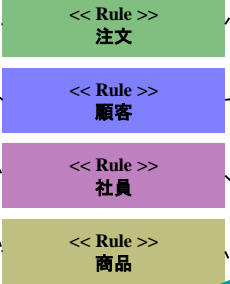
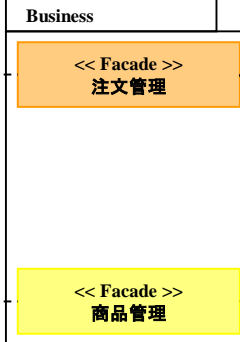
顧客
テーブル

社員
テーブル

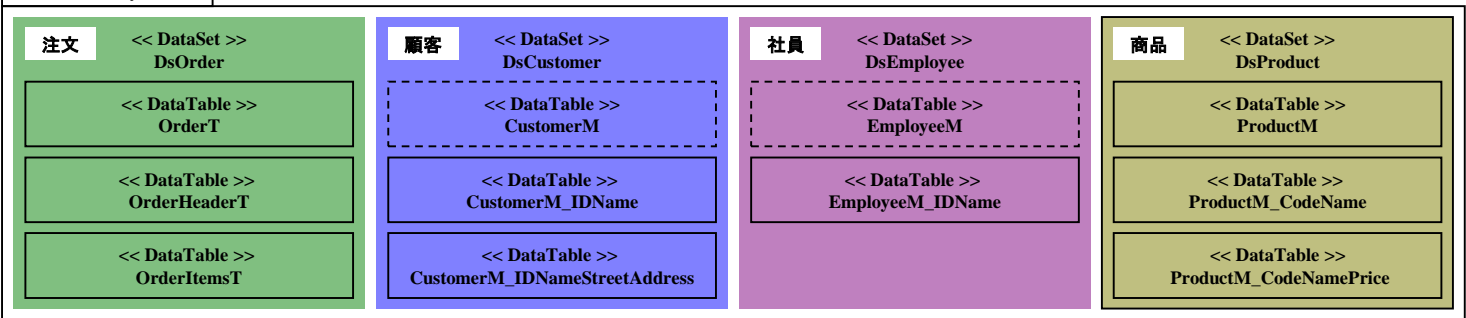
商品
テーブル

なぜ？

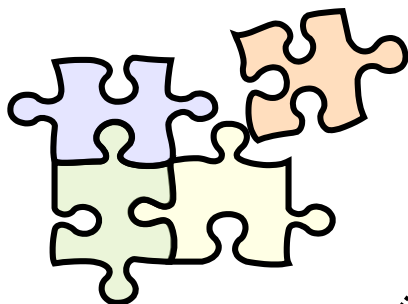
注文DataAccessから「顧客」「社員」「商品」テーブルを参照している。
→RuleやFacadeで結合するよりも、SQLで結合させた方が性能的に有利なため(更新は許さない)。



BusinessEntity



業務要件と実装の関連



「注文管理者 (アクター)」が「注文を管理する (目的レベルのユースケース)」のために、注文管理クラス (Facade) を作成する (画面がWindowsFormでもWebFormでもWPFでも共通)。
 「注文管理者」とシステムのやりとり (アクターが注文番号を指定して検索すると、システムは対象の注文データを画面に表示するetc)は、「注文...画面」で行い、1回のやりとり (トランザクション制御や各やりとり固有の業務ロジックも含む)は、注文管理クラスの1メソッドで対応する。
 RuleはどのFacadeから利用されるかは意識しない (依存しない) ようにする。
 DataAccessはRuleに対してEntityの永続化手段を提供する (物理的DB配置や永続化手段は隠蔽)。

注文を管理するための画面

- << UserInterface >>
注文一覧画面
- << UserInterface >>
注文登録画面
- << UserInterface >>
注文更新/削除画面

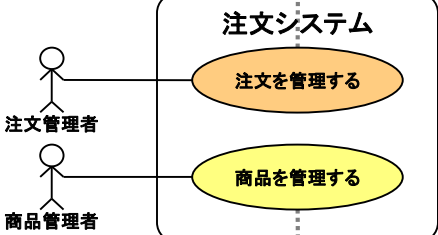
1..*

1

<< Facade >>
注文管理

ユースケースシナリオ
(サービスのワークフロー)

ユースケース図 (ユーザ目的レベル)



注文システム

注文を管理する

商品を管理する

注文管理者

商品管理者

商品を管理するための画面

- << UserInterface >>
商品一覧画面
- << UserInterface >>
商品登録画面
- << UserInterface >>
商品更新/削除画面

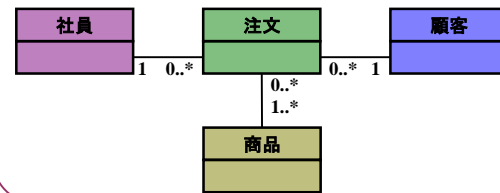
1..*

1

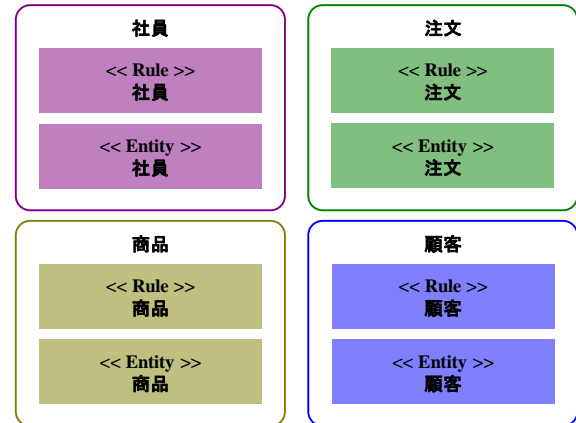
<< Facade >>
商品管理

ユースケースシナリオ
(サービスのワークフロー)

クラス図 (概念レベル)



注文システムのクラス (テーブルモジュール)



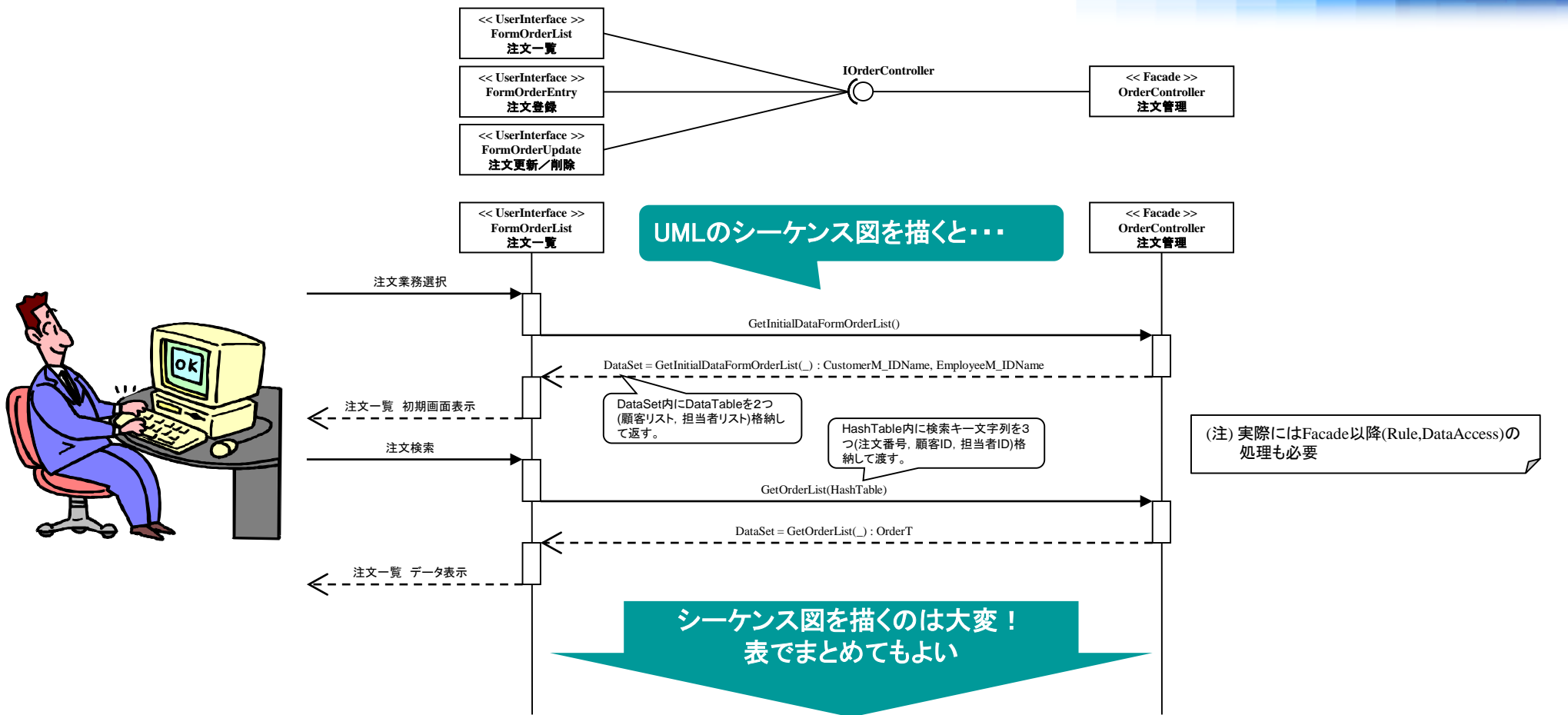
純粋なオブジェクト指向のドメインモデルではなくテーブルモジュールとし、Rule(操作)とEntity(データ)をクラスとして分離する。
 例えば「注文」の場合、「注文ヘッダ」や「注文明細」のEntityのデータ件数に関わらず「注文」のRuleのインスタンスは1つとする。
 ただし、注文する商品によって税計算が異なる場合等、「継承」「委譲」「各種デザインパターン」等、オブジェクト指向の利点を活かす設計を目指す。

画面イメージ/画面項目説明



Facadeインターフェース仕様

UserInterface～Facadeの相互関連図(シーケンス図)



Facade Interface	Facadeへの引渡情報(パラメータ)	Facadeからの返却情報(戻り値/OUTパラメータ)	説明
注文一覧画面 初期情報取得 GetInitialDataFormOrderList()	なし	戻り値：初期情報(DataSet) 顧客リスト CustomerM_IDName 担当者リスト EmployeeM_IDName	画面初期表示用のデータを取得
注文一覧画面 注文ヘッダリスト取得 GetOrderList()	注文データ検索条件(HashTable) 注文番号(String) 顧客ID(String) 担当者ID(String)	戻り値：注文ヘッダリスト(DataSet) 注文ヘッダリスト OrderT	検索条件と一致するデータを取得

(注) マルチスレッドでの制御系システムの場合は、シーケンス図での表現が望ましい。



End
End