株式会社ロングタームシステムマネジメント

フレームワーク仕様概要 (ドラフト版)

（フレームワーク開発コード：アストラル・アパッチ）

アストラルとは人智学。人智の集合体。
アストラル・アパッチは文化、概念の集合体であり、
技術の集合体（Struts、Seasar ）とは異なるレイヤーに存在する。

2011年02月01日

LTSMフレームワークチーム

目次

1. 導入実績（2）

- 中央官庁　補助金・委託費・運営費交付金管理システム　360人月
  - 工数圧縮率　150人月÷12人×1.5ヶ月×稼動率2倍（月間300時間労働）× 100% = 416%
  - 前提条件、Java経験10年以上、MVCモデル5年以上、アストラルアパッチ1年以上の技術者を使用

約**1／4**に工数を圧縮し

コストの**3倍の利益**を実現。

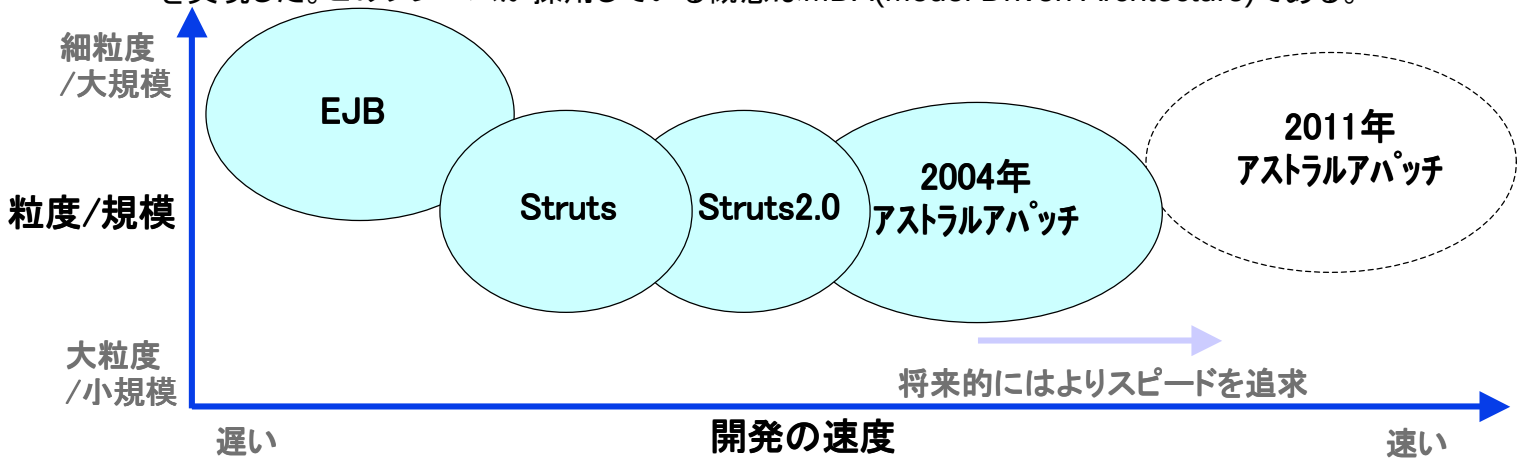
1. 導入実績（3）

- 独立行政法人　補助金・委託費・運営費交付金　契約・交付管理システム　120人月
  - 年間3000億の予算管理　現在最新版（120人月）稼働中

- 中央官庁　補助金・委託費・運営費交付金管理システム　360人月
  - 年間2兆円の予算管理　現在最新版（360人月）稼働中

2. フレームワークの3つの誕生背景（1-1　スピード重視）

- 実装のスピード
  - ユーザーニーズ
    - 弊社は9年間、大規模独立行政法人、中央官庁の基幹系、情報系システム開発に携わってきた。その間一貫したユーザーニーズは、1分でも早く機能を開発し運用に乗せることであった。このため、市販のフレームワークでは妥協のない速さへのニーズに応えることはできず、新たなフレームワークを開発する必要があった（2004年のこと）。
  - 速度と規模の両立
    - フレームワークに要求される機能は、EJBのような大規模前提で実装に時間がかかる方式より、小規模から大規模まで使え、イニシャルランニングの速さと、仕様変更対応の速さを備える必要があった。
  - JAVAプログラムの量の軽減
    - 上記のような速さへの要求を実現するため、このフレームワークはシステムの記述言語にXMLを採用し、JAVAプログラムの量を減らす事で、モレ、ミスなどのバグを減らし、品質を向上し、開発速度を上げる事を実現した。このフレームが採用している概念はMDA(Model Driven Archtecture)である。



2. フレームワークの3つの誕生背景（1-2　スピードと可読性）

- 可読性（解析性）の高さ
  - プログラムは小説だ
    - プログラムとは、いわば小説である。宣言があり、処理があり、戻り値のある、文章である。インターネット以前のシステムは、1つか2つの言語で記述され、文章としてプログラムを読む事ができた。
  - 複雑化するシステム
    - インターネットの出現により、使用する言語は最低2つ以上となった。具体的には、クライアントはブラウザー（ HTML ）と、サーバサイドの言語である。HTMLにはJavaScriptのようなスクリプト言語を含めることが多い。サーバサイドの構成は、オープンアーキテクチャーであるJAVAをベースとしたフレームワーク、または、マイクロソフトのドットネットフレームワークの採用が主流となっている。クライアントとサーバでは言語構造と呼び出し関係が大きく異なり、2つを1つの文章として読む事は難しい。
  - JAVAプログラムを減らす
    - インターネットの出現は、システムの適用範囲を大きく広げたが、同時に複数の技術でシステムを構成する事を要求した。 システム開発の前提条件が複雑になった以上、習熟度を上げても、以前のような可読性（解析性）の高いプログラムの製造は難しい。上記のような混沌とした状況を解決するため、このフレームワークはシステムの記述言語にXMLを採用し、JAVAプログラムの量を減らす事で、可読性を上げることを実現した。
- 設計書とプログラムの同期
  - 設計書とプログラムを一致させる
    - オープン系システム開発では、設計書とプログラムは一致しない。設計・製造期間が短い、仕様変更時に設計書を直さない、スキル不足の技術者が多いことなどの原因が上げられるが、期間やスキルの問題を今すぐに改善できるものではない。よって、スキルではなく、技術により、設計書とプログラムを同期させる機能を実現した。（独自のMDA(Model Driven Archtecture)エンジンを開発。）

2. フレームワークの3つの誕生背景（2-1　お客様重視の最適化）

- 最適化と寿命
  - お客様にとって重要な事は結果である
    - お客様にとって最も重要な事は3つ、予算内、期間内、求める品質の実現だけである。つまりどのような設計方法、製造方法、実現の方式かは重要ではなく、求める業務に沿った機能、予算内、期間内、求める品質を実現することだけが重要である。
  - どのレイヤーを最適化するか
    - 組織論の問題であるが、目的を達成する方法を、どのレイヤーで最適化するかによって結果は変わってしまう。例えば、経営層なら、社員の生活を守り、給与をより良く、雇用を促進し、税を払い国益を守る。これを実現するには、利益が最優先事項となる。事業部単位では自事業部の利益を最大化することに注力する。この場合、他の事業部との最適化が、はかれない場合もある。社員単位では、自分に与えられた仕事を行う。しかし、場合によっては、個々の層で保身に走り、新たな概念や変革を受け入れず、利益を最大化できないこともある。アストラル・アパッチは、上記の予算内、期間内、求める品質の実現と利益の最大化に最適化されている。つまり、SIの技術指向を無視し、お客様利益を最大化するフレームワークである。（SIの技術指向とはJava、Struts、EJB、その他現在までに発生した様々な言語や複雑な技術を指す。システムとは、お客様の目的と結果が一致すれば、どのような実現方法でも構わないというのが弊社の考えで、本来はCOBOLのような生産工学に基づく言語こそ素晴らしいという考えである。）
    - アストラル・アパッチは、既存の概念を否定し、お客様の予算、納期、品質を守り、SI事業者に利益をもたらす。しかしながら、お客様に、その存在が知れてしまった場合、SI事業者は値下げの圧力を加えられる可能性がある。つまり、SI事業者の利益を最大化するということに最適化できない可能性がある。
  - 技術の寿命
    - 技術には寿命がある。メインフレーム（COBOL）全盛期、ワークステーションとPCのクライアントサーバ（VC、VB）、WEB（Java、PHP）、それぞれ優れた特性をもっているが、どの技術が旬であるかは、常に異なる。技術には寿命があり、言語より、フレームワークが陳腐化するのは早い。よってこのフレームワークを世に送り出すことは、弊社内では考えていなかった。

2. フレームワークの3つの誕生背景（2-2　お客様重視の最適化）

- 採用権の委譲
  - お客様からの技術採用権の委譲
    - アストラル・アパッチは、世に知られていない独自のフレームワークである。その実装方式、XML定義の意味、Javaクラスの構成、全てが公開されていない。そのようなフレームワークを採用するためには、お客様から、技術の採用権（決定権）を全て委譲していただく必要がある。つまり、メーカー、SI等の下請けとしてでは、アストラル・アパッチは使用できない。

  - 弊社の業務範囲とシステム開発の考え方
    - 弊社のシステム開発は、経済産業省及び、その配下の独立行政法人の仕事しかおこなわない。特に、補助金、委託費、運営費交付金の管理業務に特化し、それ以外の業務範囲の仕事はあまり行わない。民間等のシステム開発は業務範囲に含まれない。よって、アストラル・アパッチを使用できない開発は行わない。弊社が経済産業省及び独法の予算管理に特化する理由は、業務が法律で決まっているためである。通常の要件定義は民間企業それぞれの業種・文化から生まれた業務が存在し、システムはその業務に沿わなくてはならない。業務自体は民間企業にしか解らない。よって要件定義は混迷する。しかし、中央官庁及び大規模独立行政法人は財政会計六法に従う義務があり、それを逸脱する事はできない。つまり、財政会計六法こそが業務であり、お客様からの要件定義時に、弊社が財政会計六法の補助金・委託費・運営費交付金の知識があれば、要件のモレ、食い違いを防ぎ、高精度な要件定義を完了する事ができる。（弊社は、システム開発に他に、グーグル特許を分析し利用したSEO業務、自社パッケージの開発、運用業務がございます。）

2. フレームワークの3つの誕生背景（3-1　人材とオフショア）

- 定量化と品質（人月という束縛）
  - お客様の利益を守る
    - 弊社の社風は社員を大切にすることである。しかしながら、他の零細企業ではどうなのだろうか？エンジニアを派遣の商品としてしか考えておらず、人月という非人間的な考えに縛られて能力を正当に評価しない、また能力の育成に資金使わない会社も多いのではないだろうか？そのような会社が、開発スタッフとして入り込んでいる場合、我々はどのようにしてお客様と自分の利益を守るのだろうか？

    - 某メーカー様の仲間を尊敬するというコンピテンシー（行動特性）を、そのまま当てはめる事ができるなら、弊社のフレームワークは必要ないのかもしれなし。しかし、それは、あくまで個々が善意に基づきプロジェクトに自発的に貢献することが前提条件である。そのような人材とは異なる人を使う場合、予算、納期、品質を守り、かつ利益を確保するためには、誰が製造しても、同じ品質になり、製造を定量化できることが必須条件である。

    - アストラル・アパッチのコンセプトは沢山あるがその中に、製造の定量化、誰が製造しても、同じ品質である、が上げられる。アストラル・アパッチはXML定義情報とJavaソースコードを読み込み動くエンジンだが、記述の自由度はJavaソースコード部分に限られている。しかも、 Javaソースコード部分のメソッドの呼出順序などはある程度決められているので、だれが記述しても同じようなソースコードになる。つまり、可読性が高く、同じ品質、製造を定量化できるという特徴がある。別の角度から見ると、アストラル・アパッチは自由にスクラッチでソースコードを記述できる部分を、極端に減らしたので、概念としては開発者の創意工夫の権利を許していない。つまり、新たなイノベーションが生まれないということになる。

2. フレームワークの3つの誕生背景（3-2　人材とオフショア）

- 国内の産業を守る
  - オフショア開発との戦い
    - 数年前まで、中国、インド、最近では、ベトナムなどのオフショア開発が流行っている。しかし、我が国、日本の産業の1つである、ソフトウェア開発はこれで良いのか？上流工程だけ行い、後の製造はオフショアに丸投げしてしまえば、たたき上げで製造工程から設計、上流工程へとスキルアップしていくエンジニアは存在しなくなってしまう。つまり、製造部分はブラックボックス化が進む。また、多くの人員を必要とする製造部分をオフショアに出すという事は、若者の雇用の促進を阻むということに繋がる。

    - アストラル・アパッチは、オフショアに、コストで負けない、品質でも負けないことを目指して開発された。実際に、現在のアストラル・アパッチは、中国、インド、ベトナムなどアジアの諸国でシステムをオフショア製造した場合と比較し、コスト、品質で負けない成果物を生産することができる。

  - 余談。税金が投じられる国の仕事での矛盾。
    - 国のプロジェクトは税金を使って行われるものである。しかし、1次受けの企業が、製造工程をオフショアに出すと言うことは、税金を使って、他国の利益に貢献するということになる。一昔であれば、道路、ダムの整備で大量の日本人労働者を雇用し、生活を守る事に税金が使われていた。一次的目的は不要な物だったかもしれないが、副産物としての大量の雇用を生み出すと言う部分では成功しているといえる（しかし、そのような場当たり的な雇用を生み出すための税金の使い方は認めるべきではないと個人的には考えている）。購買力平価の低い国へ、単純労働が流れてしまうというのは、経済的概念から見れば普通のことで、避ける事はできない。しかし、私達は日本人であり、ソフトウェア産業に従事している。アストラル・アパッチはあえて、オフショア開発を否定しよう、日本人の労働を守れるか闘ってみようというところからスタートしている。

2. アストラル・アパッチの問題点（4）

- アストラル・アパッチの問題点
  - ドキュメントの不足
    - アストラル・アパッチは、100%Javaのソースコードで記述されている。特にトリッキーなコードはなく、可読性の高いフレームワークと言える。弊社は10数名のエンジニアしかいない零細企業である。エンジニアはシステム全般に精通しオブジェクト指向をよく理解している。アストラル・アパッチの全機能をドキュメント化するより、口頭及びサンプルとソースコード、Javadocで理解する方が効率的であったため、ドキュメントが少ない。このため、他の企業、他のエンジニアに説明するためのドキュメント類が不足している。
  - 余剰人員の不足
    - 弊社は零細企業であるため、大手企業のように問題点が発生した場合、窓口で一次切り分けするような体制がない。エンジニアはそれぞれのお客様があり、お客様の要求に責任を持ち、効率的に実行することを第一義としている。その為、自分のお客様以外の要求に対する対応は遅い。中規模以上の企業にあるような余剰人員はいないため、窓口業務に弱い。
- 弊社の風土
  - 弊社はお客様に、ウォルトディズニーのような素晴らしい楽しい未来を提案し、リッツカールトンのようなサービスを提供したいと常に考えています。エンジニア自身がお客様の相談相手と考え、お客様や仲間を尊敬する風土です。エンジニアというよりエンジニアの技術を持ち合わせたコンサルタント（コンサルタントSE）と言った人材で構成されています。

2. アストラル・アパッチ　余談（5）

- 余談（ご興味があればお読みください。）
  - 余談（戦略と戦術）
    - アストラル・アパッチは戦術レベルの開発支援クラス群であり、戦略レベルの状況をかえるような力はない。つまり、始めから明らかに無理な開発を、可能にしてしまうほどの力はない。
  - 余談2（悲しい現実）
    - 某大手コンビニエンスストアのシステム開発であるが、詳細設計書に間違いがあっても修正しないことになったそうだ。当然お客様は知らない。あくまで、現場のエンジニアから聞いた話で事実確認はしていない。結果システムが正しく動けばいいそうだ。このような開発は後々禍根を残す、品質偽装である。私に相談してきたエンジニアは、「これは大人の選択というもので、受け入れるべきなのだろうか？」と悲しい質問をしてきた。当然、「品質偽装だ、受け入れるならキミのエンジニアとしての矜持は失われる」と答えた。わが国で行われている開発でこのような状態のプロジェクトは無数に存在するだろう。この業種に携わるエンジニアとして、とても恥ずかしく悲しい。弊社はお客様に、ウォルトディズニーのような素晴らしい楽しい未来を提案し、リッツカールトンのようなサービスを提供したいと常に考えている。技術力はあって当然という考えに基づき、MDA（ Model Driven Archtecture ）が今は最善の選択肢と考え採用している。

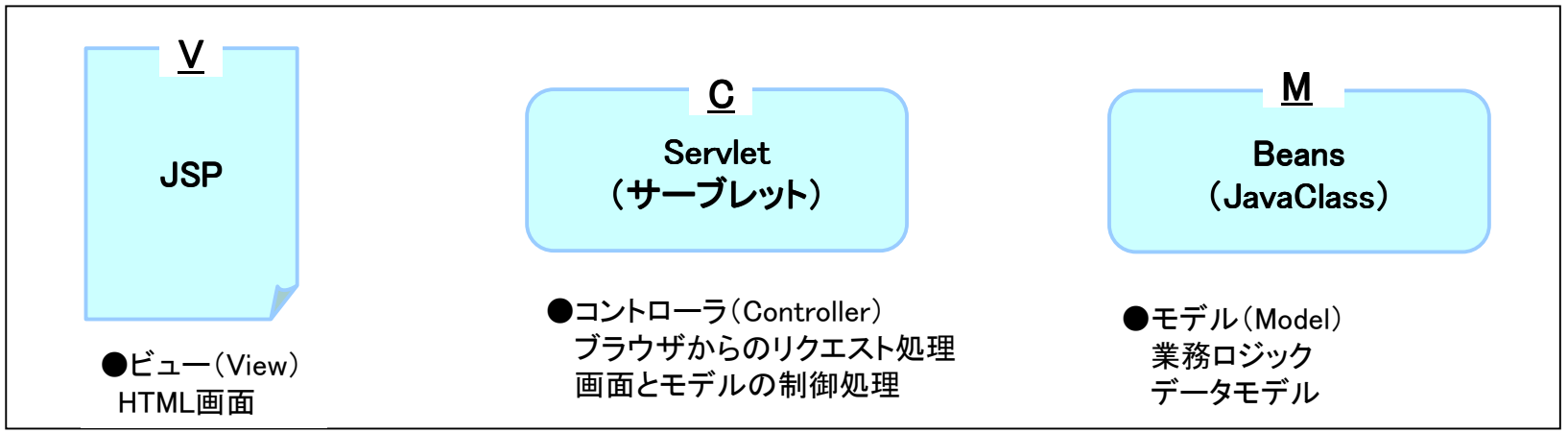
3. MVCモデル2の説明（1）

MVCモデル2を拡張したフレームワーク

- デファクトスタンダードであるStrutsに合わせたクラス構成
  - 要員の確保がし易い
    → Strutsにクラス構成を合わせることで、開発初期のオーバヘッドを低減可能
  - 要員の育成が比較的容易
    → Strutsに関する書籍は数多く入手可能であり、また研修コースも多く開かれている
- 軽量フレームワーク
  - 自由度が高く、汎用性/拡張性の面で多くのアプリケーションに対応可能
  - 軽量な分、フレームワークでカバーされない機能の作成が必要
- オープンソース ※オープンソースにする予定。現在検討中。申請式の無償利用は現在も可能
  - 特定のベンダーに依存しない
    → 移植性が高い
  - 無償で利用可能
    → 開発コストの削減

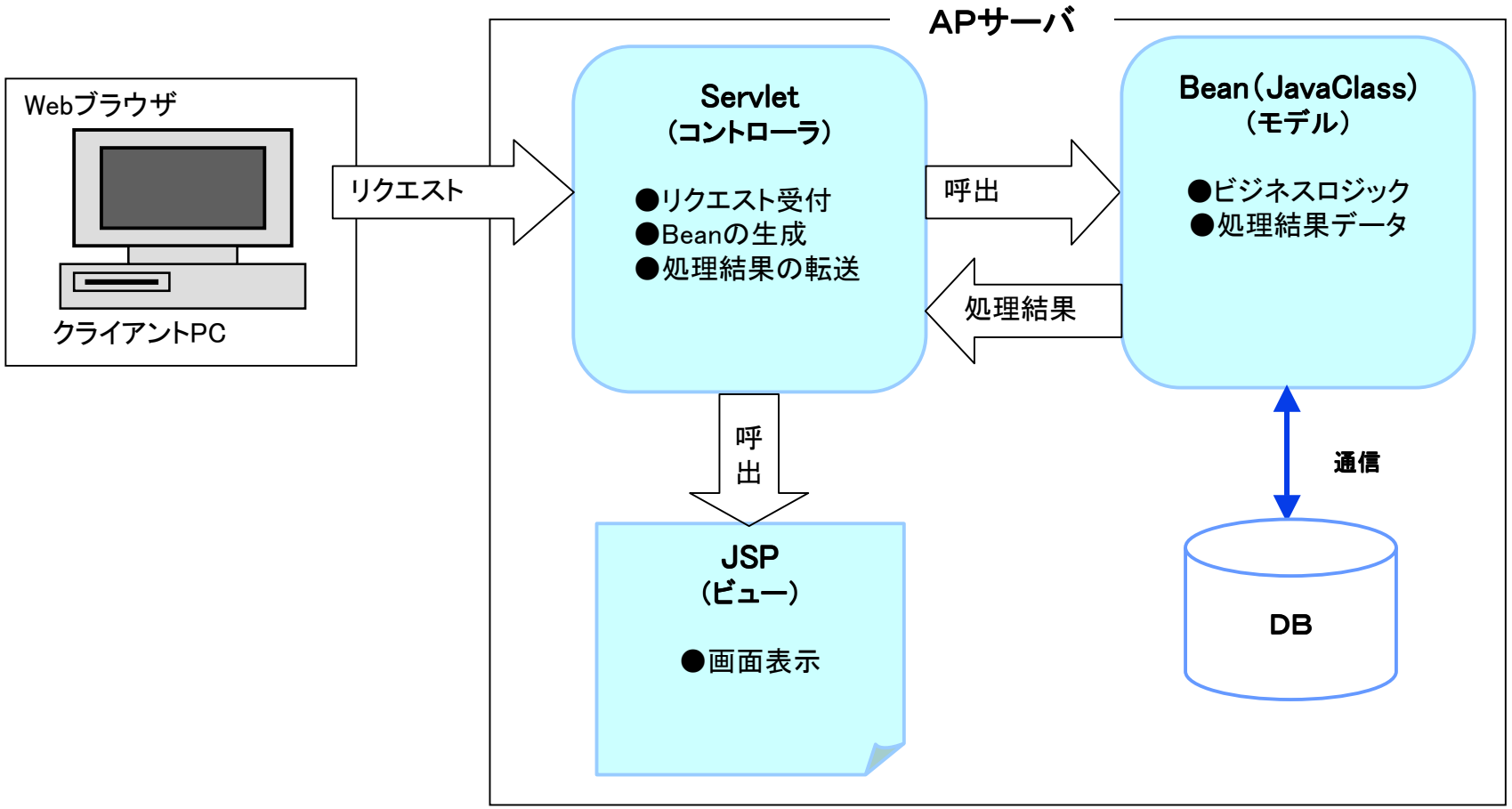
3. MVCモデル2の説明（2）

- 参考資料）MVCモデル2とは
  - JSP ＋ Servlet ＋ Beans（JavaClass） で構成
  - ブラウザからのリクエスト受付と、各オブジェクト間の制御を行うコントローラの役割は、Servletが担う
  - コントローラと、画面デザインと、業務ロジックを分離しているため、再利用性と保守性に優れたモデルである
  - JAVA Webシステム開発ではデファクトスタンダード
    - → 開発経験を持つ技術者の確保がし易い
    - → 要員の育成が比較的容易



3. MVCモデル2の説明（3）

- 参考資料）MVCモデル2の処理構造



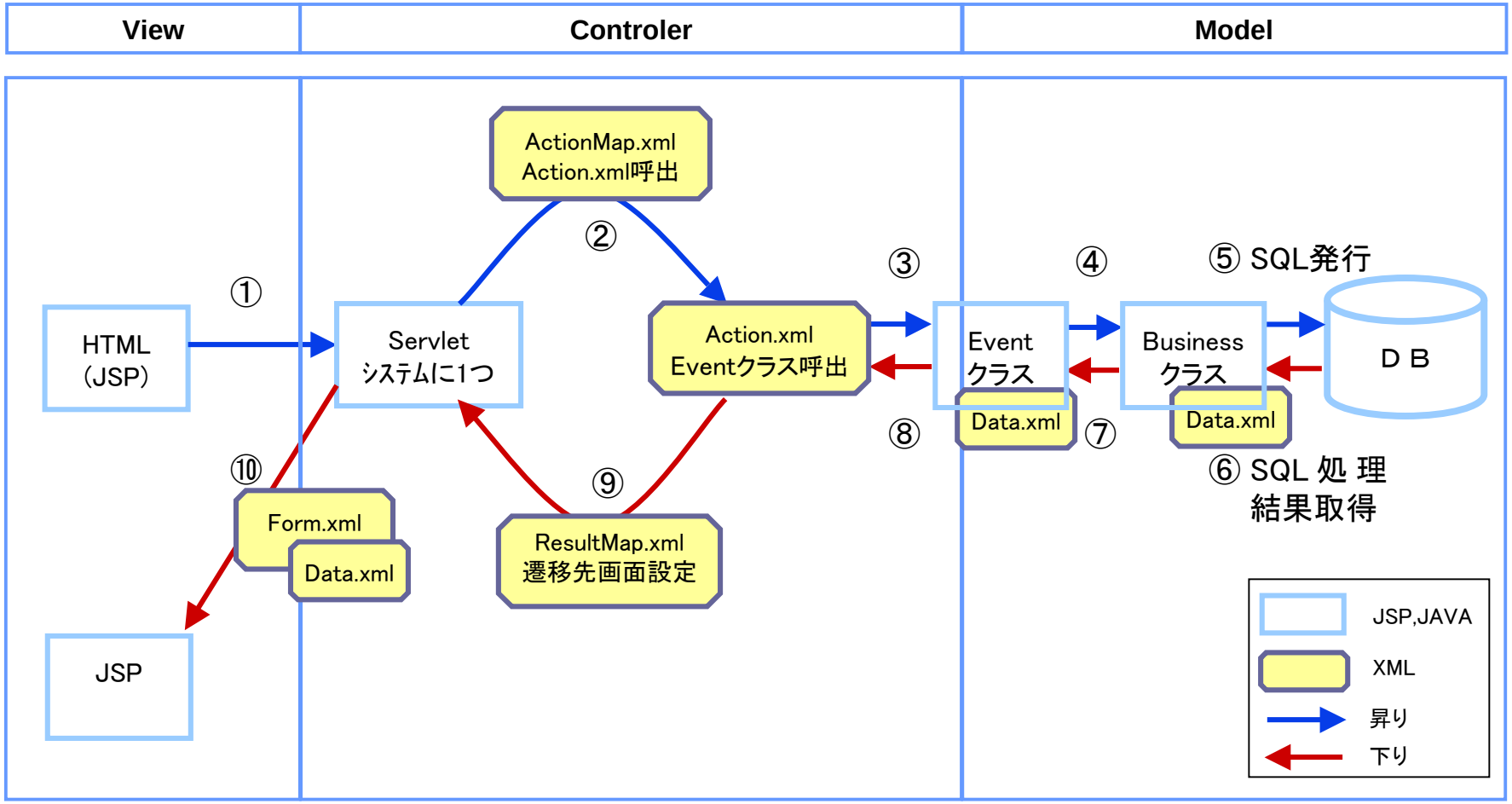
4. アストラル・アパッチの提供する機能（1）

- プログラムをXMLで記述
  - アストラルアパッチは、設計情報からプログラムを連動されるModel Driven Architecture（以降MDA）の概念を取り入れた、開発支援環境です。
  - 画面、項目、データベースとのマッピングなどの設計情報をXMLで記述することで、アストラルアパッチが設計情報を読み込みシステム稼動します。またXML設計情報から、各種設計書を自動作成することもできます。

    - ＜アストラルアパッチの特徴＞
      - 設計情報をXMLで記述することで、アストラルアパッチが設計情報を読み込みシステム稼動します。このため、基本設計フェーズからシステムのプロトタイプを稼動させることが可能です。
      - 項目の仕様変更（必須入力、レングス、属性など）はXML定義を変更するだけです。
      - 項目の追加、削除はXML定義の変更と、わずかなJavaプログラムの修正で完了します。
      - 新規画面の開発は、用意された雛形の指定文字を一括置換するだけで、画面の基本的な動作を実現可能です。
      - 設計情報を記述したXMLから、各種設計書を出力できます。

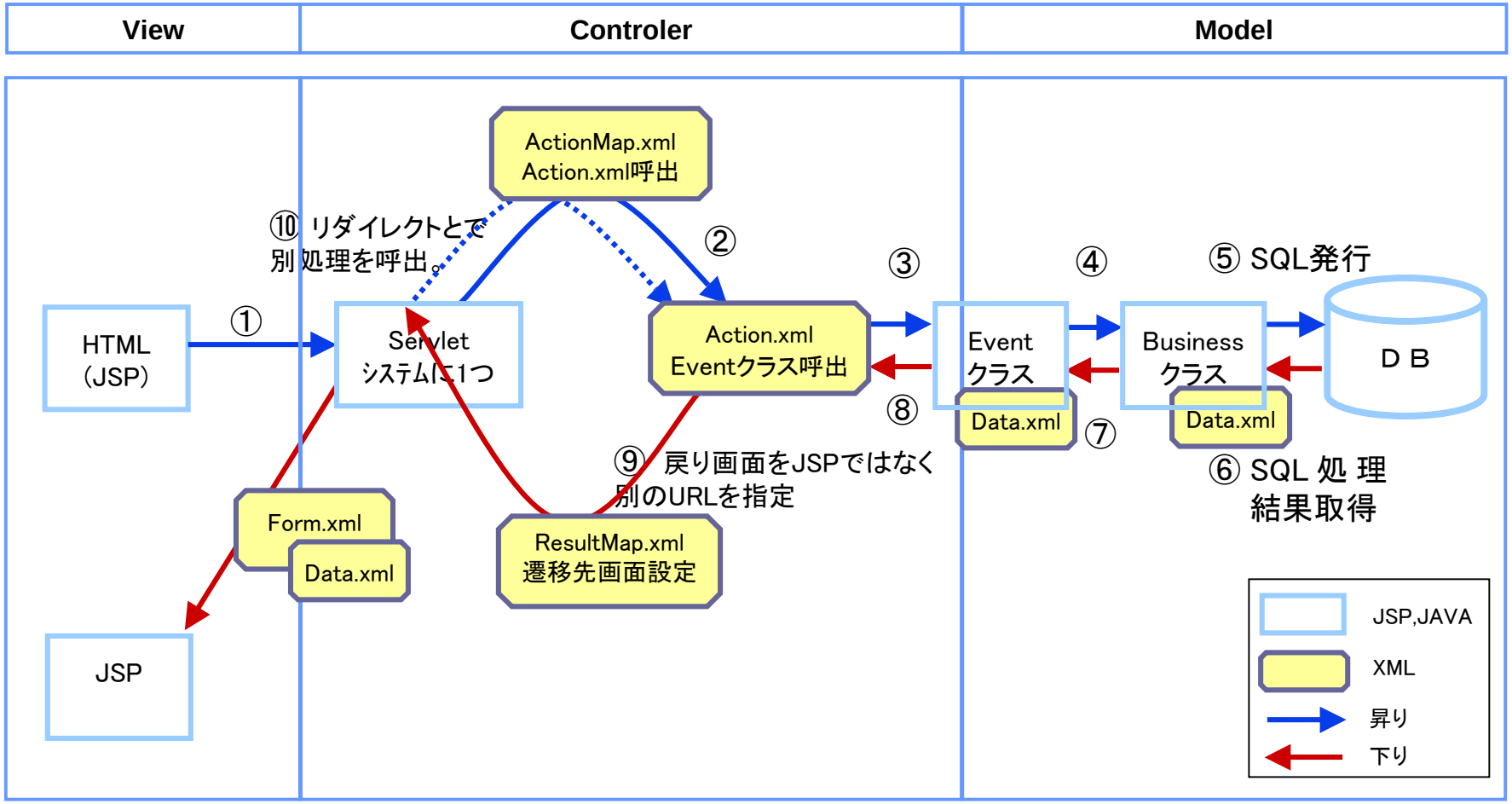
3. クラス構成図（2-1）

- アストラル・アパッチ　クラス構成図　（シンプル）



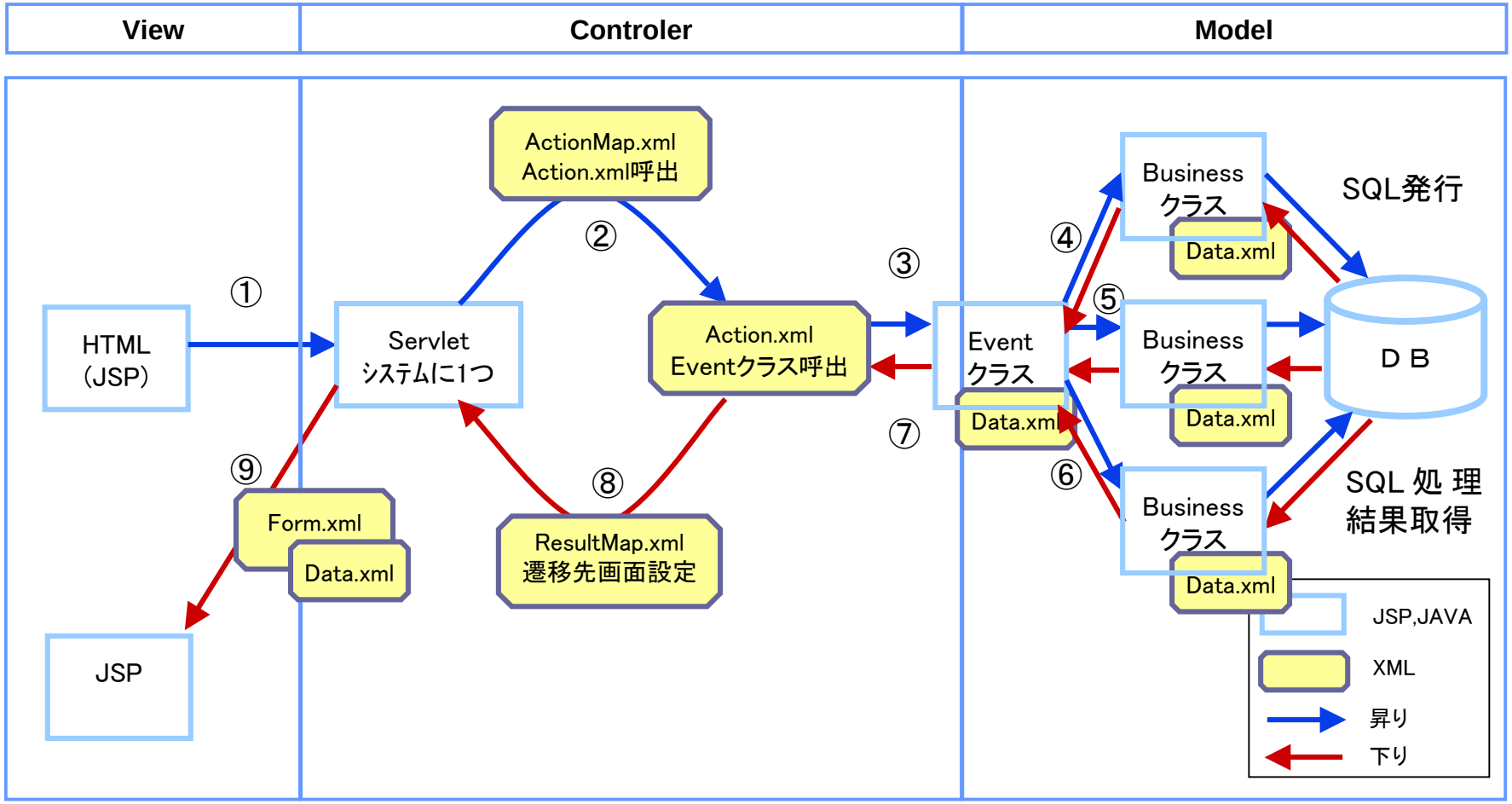
3. クラス構成図（2-2）

- アストラル・アパッチ　クラス構成図（リダイレクトで再度別処理呼出）



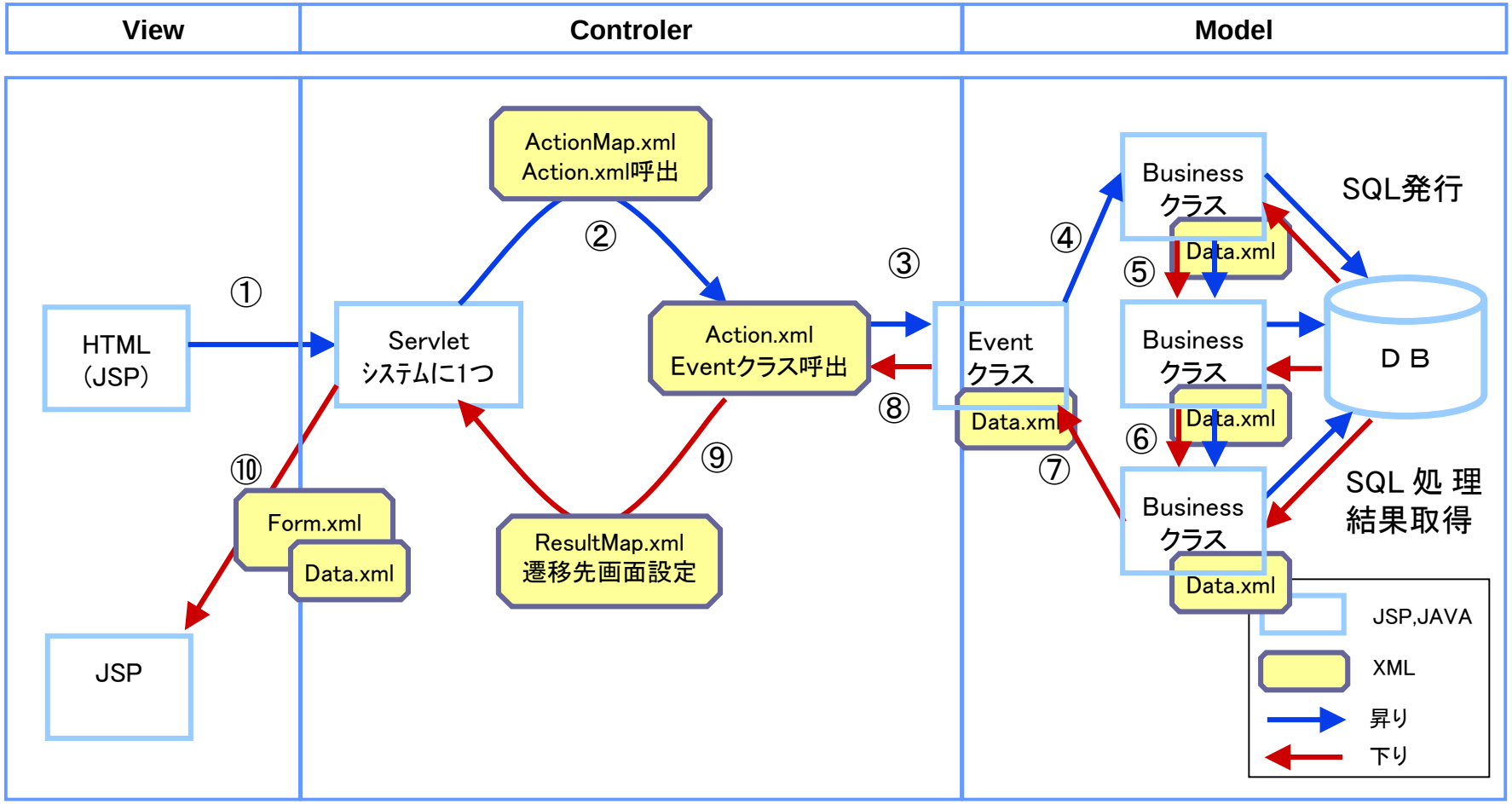
3. クラス構成図（2-3）

- アストラル・アパッチ　クラス構成図（イベントクラスから複数のビジネスクラスの呼出）



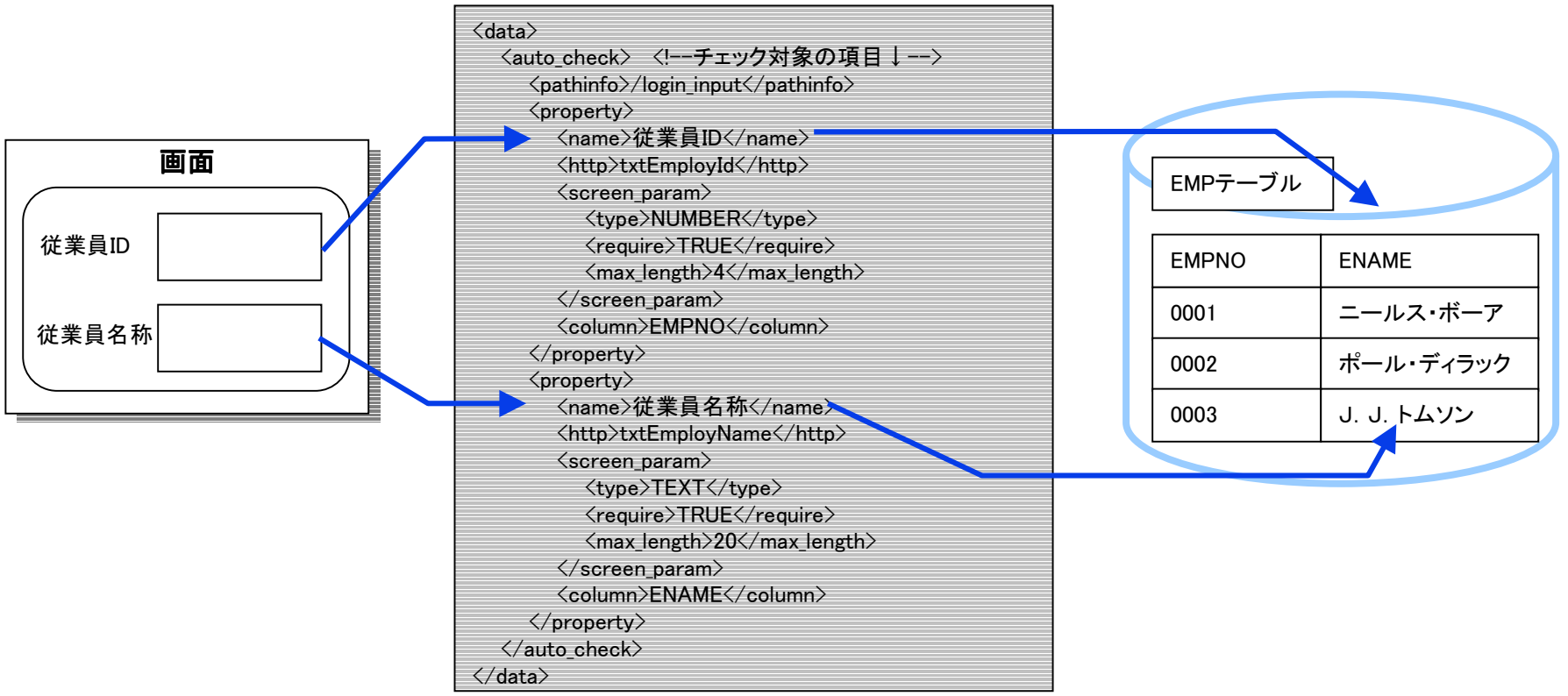
3. クラス構成図（2-4）

- アストラル・アパッチ　クラス構成図（ビジネスクラスから別のビジネスクラスの呼出）



4. 画面とデータベースのマッピングイメージ（3）

- プログラムをXMLで記述
  - 画面項目・入出力をXMLで定義
    - 画面の入力情報、データベースからの取得情報を格納するData.xmlに、項目のタイプ、レングス、コントロールなどの属性情報を定義し、画面上の項目とデータベースの項目を紐付する。



4. アストラル・アパッチのクラス構成（4-1）

■ アストラルアパッチのクラス構成

アストラルアパッチのクラス構成は下記表のとおりです。

クラス種別	タイプ	機能	命名規則
Servlet.java	Java Servlet	システム共通の処理を記述するServletClass。 MVCではServletに処理の制御を記述するが、XMLフレームワークではAction.xmlに記述するため、Servletは基本的には1つである。	任意名称 ＋ ControlServlet.java
Action.xml	制御用 XML定義	処理の制御を記述する。具体的には、 画面のボタン押下などの情報を取得し、ボタン押下時の処理の実態を記述するEventクラスの指定メソッドを呼び出す。	機能名+Action.java
Event.java	Java Class	画面上のボタン押下などで呼び出される処理の実態を記述する。具体的には、画面上の入力項目の内容取得メソッドの呼出、入力チェックメソッドの呼出、データベース更新ロジックを行うメソッドの呼び出し等の、各種の処理が記述されたメソッドの呼出を記述する。処理結果に基づき次画面（JSP）への振り分け処理も記述する。	機能名+Event.java
Business.java	Java Class	データベースからのデータの作成、更新、削除、取得等のビジネスロジックを記述する。MVCモデルのモデルに相当する。	機能名 +Business.java
Data.xml	画面等項目 のXML定義	画面項目名、レングス、属性、各種入力チェック、データベースとのマッピングなどの画面等項目に関する情報を記述する。	機能名+Data.xml

4. アストラル・アパッチのクラス構成（4-2）

■ アストラルアパッチのクラス構成

アストラルアパッチのクラス構成は下記表のとおりです。

クラス種別	タイプ	機能	命名規則
Form.xml	画面で使用するデータクラスのXML定義	画面上で使用するデータクラスを記述する。1画面に1つ作成する。Form.javaに継承させる。	機能名+Form.xml
Form.java	Java Class	JSPへのデータ値受け渡し用のクラス。Form.xmlを継承し、1画面に1つ作成する。	画面名Form.java
Jsp	Jsp	画面表示を行う。	機能名.jsp
actionmap..xml	仮想URLの定義	HTTPリクエストに呼び出される機能（クラス／メソッド）を仮想URLにする定義を記述する。	actionmap..xml
resultmap.xml	jspの仮想名称の定義	Jspの仮想呼出名称の定義.を記述する。	resultmap.xml

4. XML記述例（5-1）

- プログラムをXMLで記述
  - クラスの呼出定義
    - ActionMap.xml、Action.xml、ResultMap.xml、で定義する。クラスの呼出関係を変更する場合も、XMLを変更することで実現する。ハードコーディングの箇所が大幅に減るため、従来のフレームワークと比較し、メンテナンス性が向上する。

① ActionMap.xml ： クライアントブラウザーからボタン押下などによる、リクエストで呼び出されるMVCのコントロール部（C部）の定義

ActionMap.xml

```
<!— ログイン処理の定義 -->
<pathinfo>/login_input</pathinfo>
<actionhandle>login.action.LoginAction</actionhandle>

<!— 委員会一覧画面処理の定義 -->
<pathinfo>/committee_list</pathinfo>
<actionhandle>committeelist.action.CommitteeListAction</actionhandle>
```

4. XML記述例（5-2）

- プログラムをXMLで記述

② Action.xml　：コントロール部からビジネスロジック（M部）の呼出定義

Action.xml

```
<!-- デフォルト -->
<isClicked>
        <business method="defaultEvent">committeedetail.business.CommitteeDetailBusiness</business>
</isClicked>

<!-- 新規作成ボタン押下 -->
<isClicked>
        <pushButton>btnCommitInsert</pushButton>
        <business method="commitInsertEvent">committeedetail.business.CommitteeDetailBusiness</business>
</isClicked>
```

③ ResultMap.xml：ビジネスロジックの結果を、画面表示するためのJSP（V部）の呼出定義

ResultMap.xml

```
<!— ログインJSPの定義 -->
<result>login_input</result>
<jsppath>/login/loginInput.jsp</jsppath>

<!— 委員会一覧JSPの定義 -->
<result>committee_list</result>
<jsppath>/committeelist/CommitteeList.jsp</jsppath>
```

4. Validater（5-3）

- validater（入力チェック）をXMLで定義
  - validaterをXMLで定義する。validaterは、日付型チェック、数値型チェック、レングスチェックなど基本的なチェックを提供する（ validaterは記載内容より充実しています ）。

XML定義タグ	validaterの種類	処理の内容
Type	データタイプチェック	項目のデータ型をチェックする。 DATE ： YYYY/MM/DDの形のチェック NUMBER ： 数値型のチェック
require	必須入力チェック	必須入力チェックをおこなう。
isSymbol	不正文字チェック	不正記号(「'」,「＜」,「＞」,「"」)を文字列に含むかチェックする。
maxLength	レングスチェック	文字列が最大バイト数以内かチェックする。
minLength	レングスチェック	文字列が最小バイト数以上かチェックする。

4. アストラル・アパッチの提供する機能（6）

- アストラル・アパッチのクラスは50以上のクラス、1000以上のメソッドで構成されている。

- 以降に記載されている機能は、全体機能のほんの一端であり、全容が知りたい場合はJavaDocが整備さえている。

4. アストラル・アパッチの提供する機能（6-1）

- アストラル・アパッチがXML定義で出来る事　コントローラ[Servletクラス]

MVCc	機能	機能の内容	アストラル・アパッチでの処理方式
C [Servlet]	リクエストと処理クラス、画面遷移のマッピング	処理要求(主にリクエスト)を業務ロジックに振り分ける。業務ロジック実行結果をうけ、その後の画面先の制御する。	アストラル・アパッチでは、通常、システム全体またはサブシステム単位にServletクラスを1つにする。画面遷移はActionMap.xml、Action.xml、ResultMap.xml、で定義する。クラスの呼出関係を変更する場合も、XMLを変更することで実現する。ハードコーディングの箇所が大幅に減るため、従来のフレームワークと比較し、メンテナンス性が大きく向上する。
	例外処理	例外内容に合わせた処理を実行する。	例外処理は通常、catch (Exception e)を使用せず、Event.javaまたはBusiness.javaで定義する。エラーメッセージ、入力項目の背景色の変更、画面遷移先を決定できる。 アストラル・アパッチでは、通常Servletクラスをシステム全体またはサブシステム単位に1つにする。Servletにcatch (Exception e)を記述し、他の全クラスでは、catch (Exception e)を記述せず、throws Exception {　を記述する。つまり、1つのServletで全ての予期せぬエラーをハンドリングする集中処理方式を採用している。 この方式のメリットは、プログラマ（コーダー）のレベルが低水準でcatch (Exception e)を使いたがる場合、どのクラスでExceptionが発生しているのか解らなくなる事を100%防げる。 エラー発生時には、データの整合性を保つためのロールバックも自働で行われる。Servletクラス内でロールバック方法の細かな記述も可能。 また、納期までに原因を判明できないバグが発生した場合、Servletで最終決定を行うので、エラーハンドリングで逃げる事も可能。（※推奨はできない）
	チェック管理	ページ遷移チェック等の各チェック機能の管理を行う。初期化・実行順序・実行。	ActionMap.xml、Action.xml、ResultMap.xml、で定義する。ハードコーディングはない。

4. アストラル・アパッチの提供する機能（6-2）

- アストラル・アパッチがXML定義のみで出来る事　Business[Business クラス]

MVC	機能	機能の内容	アストラル・アパッチでの処理方式
M [Model]	データベースとのマッピング	入力項目とデータベースとのマッピングを行う。	Data.xmlまたはBaseData.xmlに定義する。 プロジェクトテーブルのプロジェクト番号とマッピングする場合。 <table>プロジェクト</table> <column>プロジェクト番号</column> <comment>プロジェクトを一意に識別する番号</comment>
	HttpのGet、Postデータを自働取得する	form.setScreenParam	form.setScreenParam(request, form.getData(“todoDetailData”));で HttpServletRequest request上に展開されているGet、Postデータを todoDetailDataに自動的に代入する。
	XMLに定義されたValidationチェックを実行する	form.autoCheckScreenParam	form.autoCheckScreenParam(request, ″TodoDetailData.xml″, ″main″, form.getData(″todoDetailData″)); form.getData(“todoDetailData”)で現在代入されたいる値を、TodoDetailData.xmlに定義されたValidationチェックを実行する。
	チェック管理	ページ遷移チェック等の各チェック機能の管理を行う。初期化・実行順序・実行。	ActionMap.xml、Action.xml、ResultMap.xmlで一連の処理を制御。

4. アストラル・アパッチの提供する機能（6-3）

- アストラル・アパッチがXML定義のみで出来る事　Business［ Business クラス］

MVCc	機能	機能の内容	アストラル・アパッチでの処理方式
M [Model]	DBアクセス(JDBC/Oracle)のコネクションプール	JDBCの簡易ラッパーおよび、コネクションプール機能。	JDBCの簡易ラッパーおよび、コネクションプール機能を提供する。

	Select文SQLの発行結果支援	Select文SQLの発行結果データ（何カラムあっても）を一行でデータクラスに自働代入	Select文SQLの発行結果データ（何カラムあっても）を一行でデータクラスに自働代入
			result = statement.executeQuery(sqlSlc);
			if (result.next()) {
			// resultのデータをクラスに代入
			data.setAllColumnValues(result);
			}

	エラー発生時の自働ロールバック	エラー発生時の自働ロールバック	エラー発生時には、データの整合性を保つためのロールバックは自働で行われる。Servletクラス内でロールバック方法の細かな記述も可能。

4. アストラル・アパッチの提供する機能（6-4）

- アストラル・アパッチがXML定義で出来る事　コントローラ[Servletクラス]

MVC	機能	機能の内容	アストラル・アパッチでの処理方式
V [JSP]	画面呼び出しの定義	Jspの仮想呼出名称の定義を記述する。	resultmap.xmlに定義する。 <!-- 制度明細 --> <result>budget_detail</result> <jsppath>/grant_jsp/BudgetDetail.jsp</jsppath>
	表示項目とデータベース項目とのマッピング	表示項目とデーベースのテーブル・項目とその属性に合わせたマッピングを行う。	画面の項目表示（例） 画面の表示モードに従い入力項目なのか、表示項目なのかを自働判別し、項目を生成する。 <%= budgetDetailForm.printInputHTML("budgetDetailData", "予算年度") %>

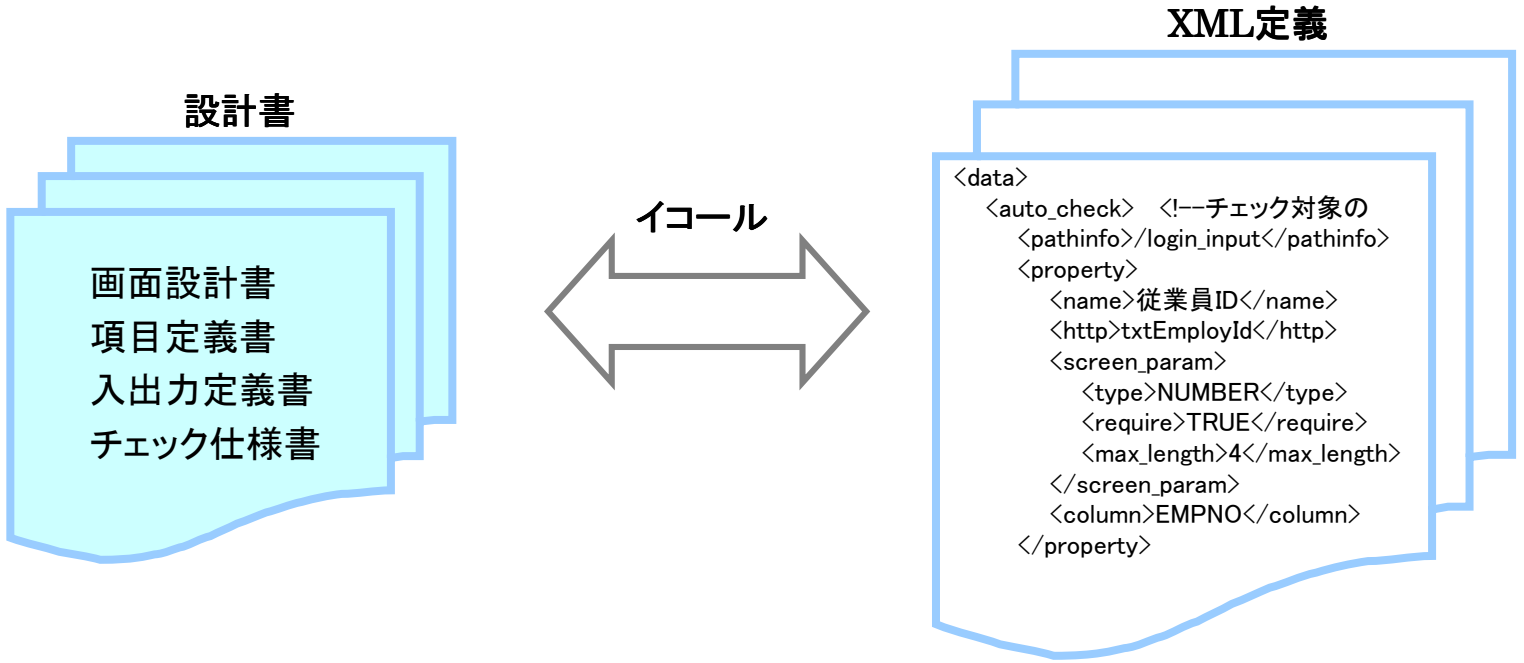
4. アストラル・アパッチの提供する機能（6-5）

- フレームワークが具備すべき機能とStruts, LTSMフレームワークの比較

機能	F/W機能としての必要性	Struts	LTSMフレームワーク	機能の内容
コントローラー機能				
リクエストと処理クラス、画面遷移のマッピング	○	○	○	処理要求(主にリクエスト)を業務ロジックに振り分ける。業務ロジック実行結果をうけ、その後の画面先の制御する。
例外処理	○	○	○	例外内容に合わせた処理を実行する。
チェック管理	○	×	△	ページ遷移チェック等の各チェック機能の管理を行う。初期化・実行順序・実行。
入出力管理				主にフレームワーク内部で使用する機能
入力データ・チェック(validater)	○	○	○	データ型、長さのチェック。
画面テンプレート	○	○	○	画面項目をテンプレートで定義する。
多言語対応(I18N)	○	○	○	ブラウザーのリクエスト・ヘッダーからロケールを取得して、メッセージのロケールを切り替えする。
サービスチェック	△	×	×	Webサービスの日々の運用時間、メンテナンス日時などのユーザーへの照会。
ページ遷移チェック	○	△	△	ページ遷移の連続性を保証する機能。
二重処理の防止	○	△	△	ボタンの2度押し防止、ブラウザーの戻るボタンの防止など。
ファイル・ダウンロード	○	×	○	サーバからファイルをダウンロードする。
ファイル・アップロードおよび版管理	○	×	○	クライアントからファイルをアップロードする。 アップロードしたファイルの履歴をとる。(版管理)
ユーティリティー				主に業務で使用する機能。
メッセージ管理	○	△	○	エラーメッセージ、確認メッセージなどの管理機能。
メール送信	○	×	○	JavaMailの簡易ラッパーを提供。
DBアクセス(JDBC/Oracle)のコネクションプール	○	×	○	JDBCの簡易ラッパーおよび、コネクションプール機能。
ログ出力	○	△	○	ログ出力。
CSV作成	△		○	CSVを作成するコンポーネント。

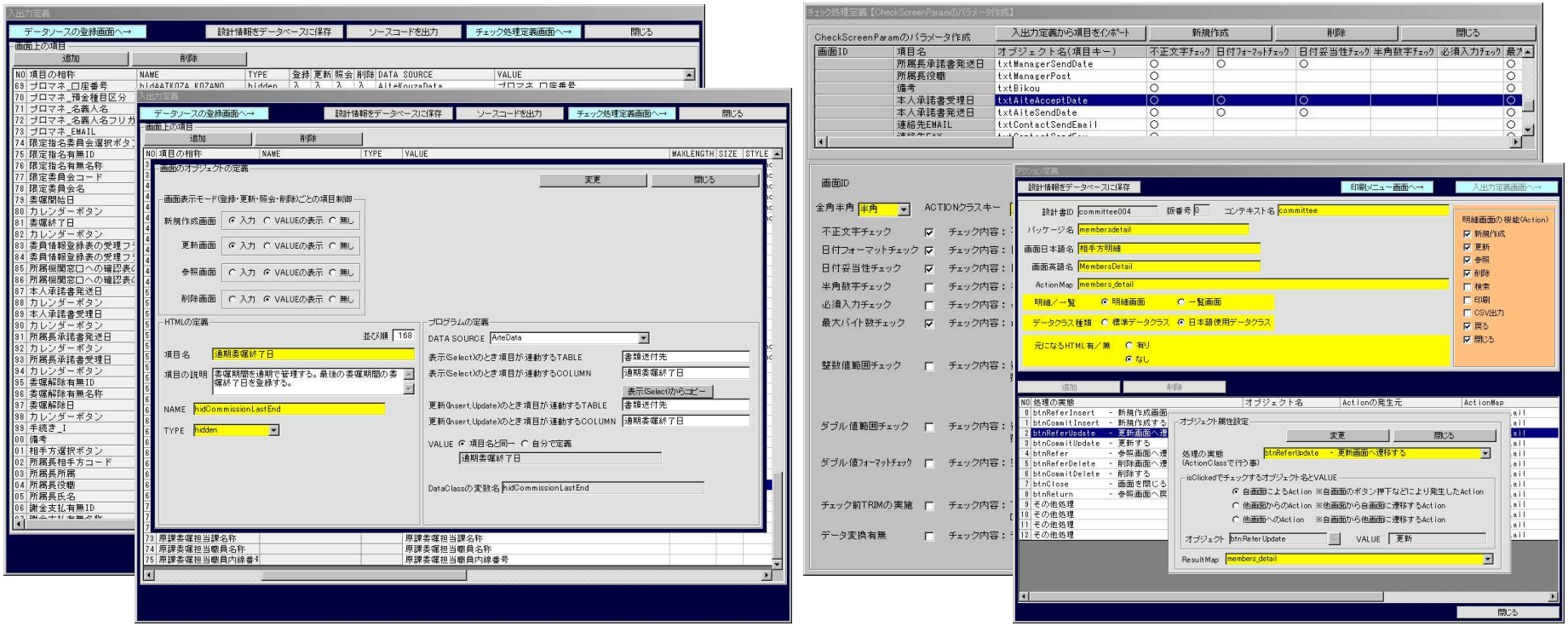
5. 設計書とプログラムの同期（1）※開発中 8月リリース

- 設計書⇔XML定義のフォワード・リバース
  - フォワード機能
    - 設計書から、ActionMap.xml、Action.xml、Result.xml、Form.xml 、Data.xml 、とBusinessクラス、JSPを出力できる。（一部開発中）
  - リバース機能
    - XMLで定義された、クラスの呼出関係の定義、画面項目・入出力の定義、チェック処理の定義情報は、専用ツールで設計書化できる。（開発中）



5. 設計書とプログラムの同期（2）

- 専用ツールで設計・製造をより効率化
  - 専用ツールで設計のミス、モレを防ぎ、設計効率を向上した。（下画面が専用ツール。最新バージョンのアストラル・アパッチではこの方式を廃止。）
  - 設計は、専用ツールの画面に入力する方法と、設計情報をエクセルファイルにダウンロードし、設計作業をエクセル上でおこない、そのファイルをアップロードする形で進めることも可能。（最新バージョンのアストラル・アパッチで採用。現在開発中）



お客様要件、品質、可読性、生産効率、メンテナンス全てを思いのままに

- 約1／4に工数を圧縮しコストの3倍の利益を実現したフレームワーク。
- エンジニアマインドから作られたフレームワークを完全否定し、生産工学に基づく唯一のフレームワーク。
- アストラル・アパッチはオブジェクト指向を否定し「MOVE 1 TO W-NO. 」の世界を肯定する。
- お客様の要求する予算、納期、品質を実現する強力なフレームワーク。
- 高い可読性から生まれるメンテナンスの容易なフレームワーク。。
- 設計書とプログラムが同期する唯一のフレームワーク。
- 進化しつづけるフレームワーク。
- アストラルとは人智学。人智の集合体。人の願い。