クラウド時代のアプリケーション開発

浅海智晴
匠Lab/edge2.cc/Javaユーザグループ
2009年9月24日

浅海のプロフィール

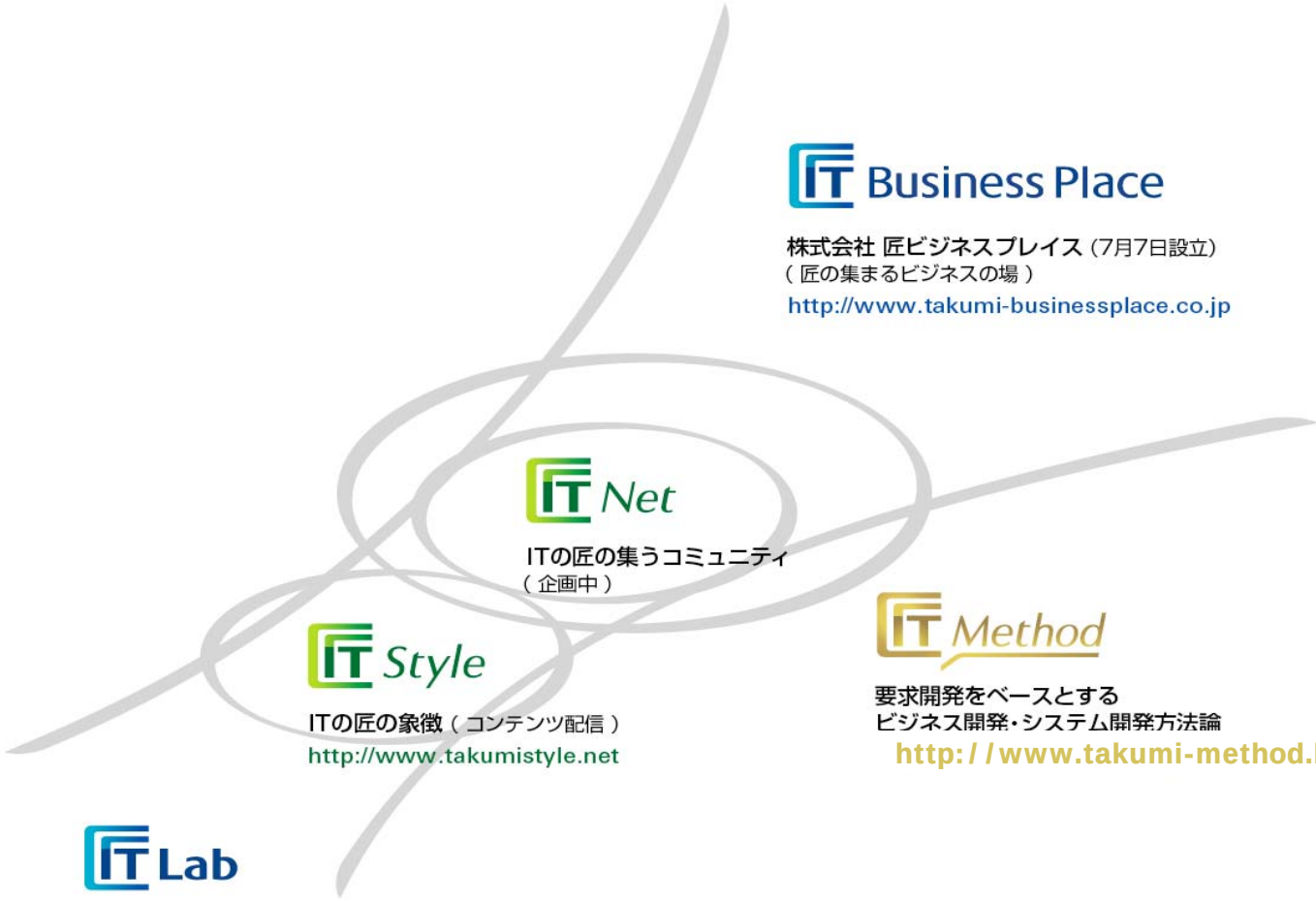
- 1985年-2001年：富士通
  - UNIX OSをビジネス向けに改造する仕事
    - ファイル管理、分散ファイルシステム、Webサーバなど
    - 信頼性、運用管理、COBOL向けの改造
  - 1993年頃からオブジェクト・モデリングの調査を始める
  - 1995年からJavaの利用を始める
  - 1998年からJava&XMLのフリーソフトを開発・公開(個人活動)
    - SmartDoc(XML文書処理系)、Relaxer(プログラム自動生成)
- 2001年-現在：浅海智晴事務所代表
  - モデリング、XML、Javaのコンサルティング、教育活動
- 2002、2003年度：IPA未踏に採用
  - Relaxer (DSLによるプログラムの自動生成)
- 2005年度-2007年度：稚内北星学園大学東京サテライト校教授
- 2007年度-現在：日本Javaユーザグループ副会長
- 2009年2月-現在：edge2.cc主宰
- 2009年5月-現在：匠Labフェロー

開発プログラム

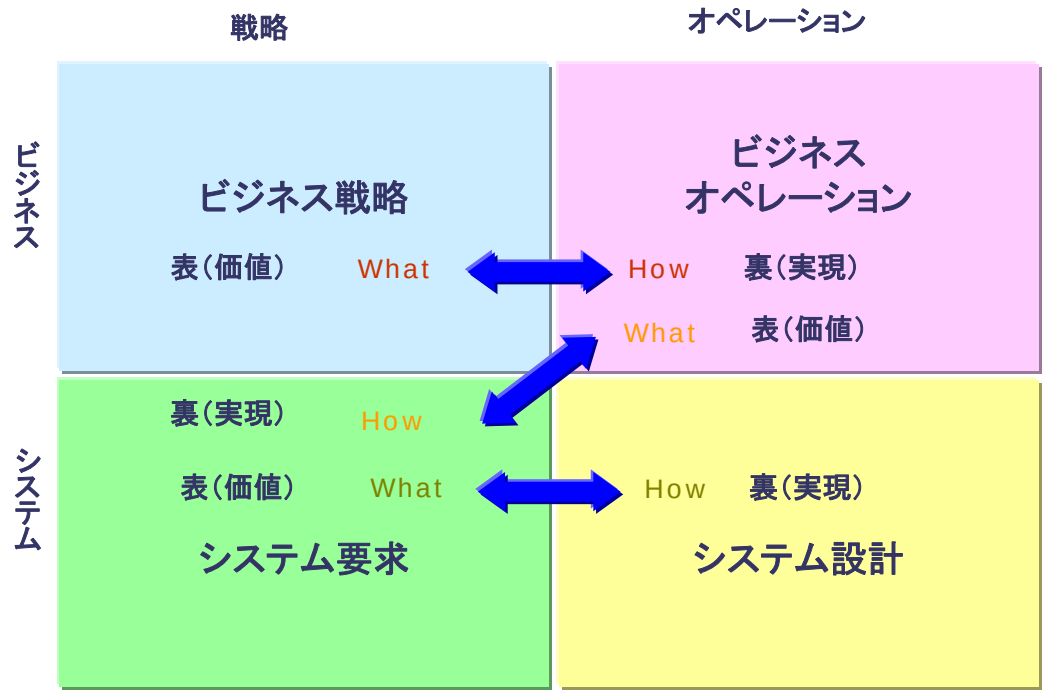
- SmartDoc (1998年)[Java]
  - XML文書処理系
  - 専用XML文書からHTML、LaTeX、プレインテキストを生成
- Relaxer (2000年)[Java]
  - XMLスキーマ言語RELAXをDSLとして用いたスキーマ・コンパイラ
  - RELAXからJavaプログラム、W3C XML Schemaなどを生成
- SmartCase (2004年、試作)[Java]
  - 専用XML文書でユースケース・モデルを記述
  - 仕様書を生成
- JavaDSL (2007年、試作)[Java]
  - JavaをDSLのメタ言語としてオブジェクト・モデルを記述
  - Javaプログラムと仕様書を生成
- SimpleModler (2008年～)[Scala]
  - ScalaをDSLのメタ言語としてオブジェクト・モデルを記述
  - Javaプログラムと仕様書を生成

SimpleModelingの本

匠グループ全体コンセプト



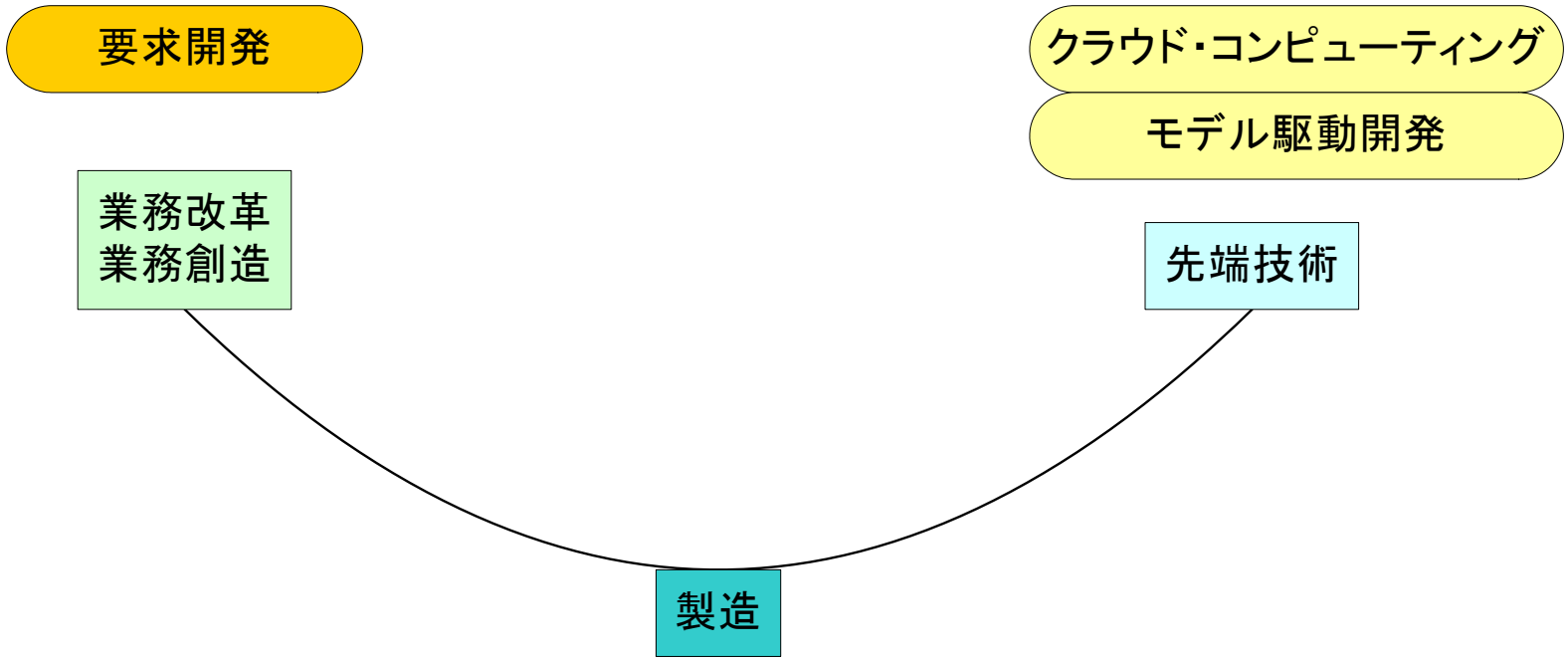
戦略には価値がなく、戦略と実現の融合に価値が生まれる



参考：　日経　Itpro 萩本・匠style研究所 「論理的美の虚像」

第7回　http://itpro.nikkeibp.co.jp/article/COLUMN/20090619/332251/
第8回　http://itpro.nikkeibp.co.jp/article/COLUMN/20090717/334022/
第9回　http://itpro.nikkeibp.co.jp/article/COLUMN/20090825/335978/

システム開発技術のスマイルカーブ



本セッションの目的

- クラウド時代におけるアプリケーション開発について考える
  - アプリケーションの構造
  - 必要とされる技術
  - 移行パス

関連雑誌記事

- 『Cloud Modeling：クラウド時代のモデリング技術』
  - UNIXマガジン 2009年春号
- 『マルチパラダイム言語Scala』
  - ITアーキテクト誌 Vol.24 (7月25日発売)
- 『クラウド時代のWebアプリ開発作法』
  - ITアーキテクト誌 Vol.25 (9月25日発売)
- 『実証研究プロジェクト「edge2.cc」の挑戦 ： アプリ開発者の目線で探るクラウドの可能性と実装手段』
  - DBマガジン誌 11月号(9月25日発売)

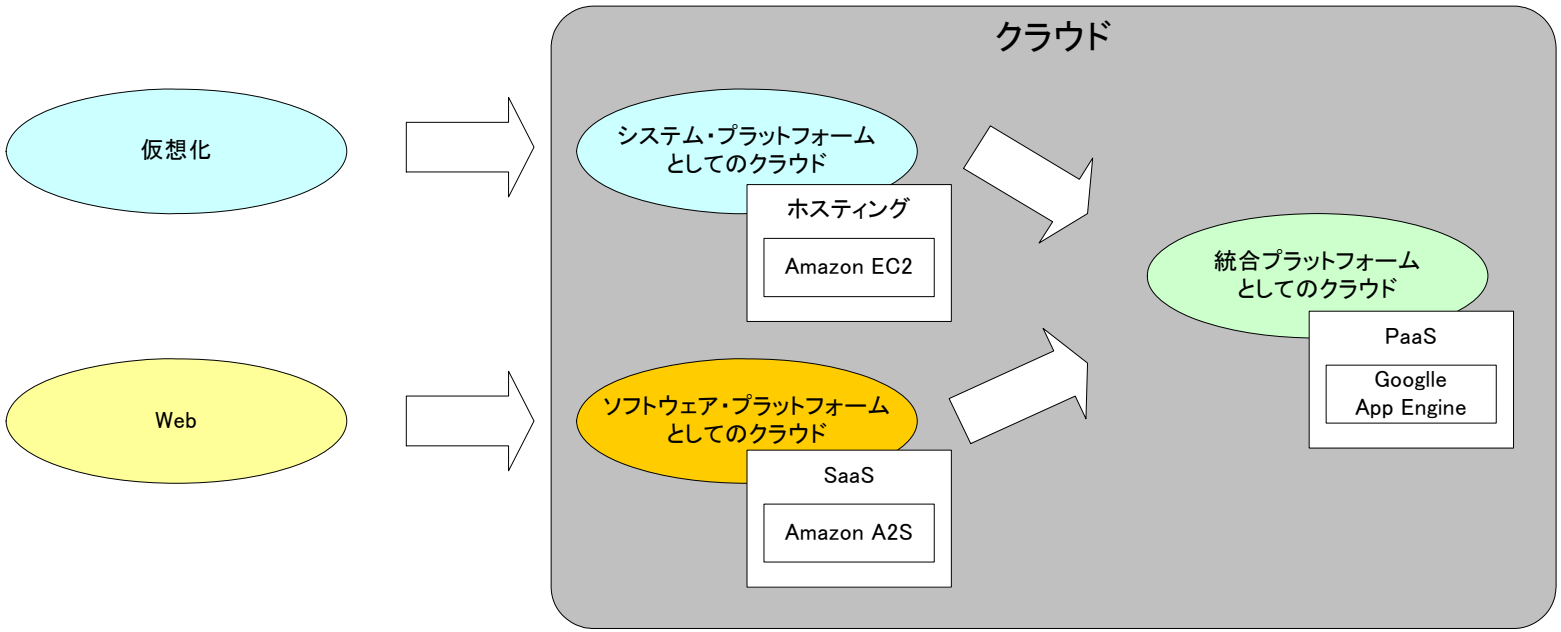
目次

- クラウド時代を確認
- クラウド時代のアプリケーション開発
- edge2.ccの活動

目次

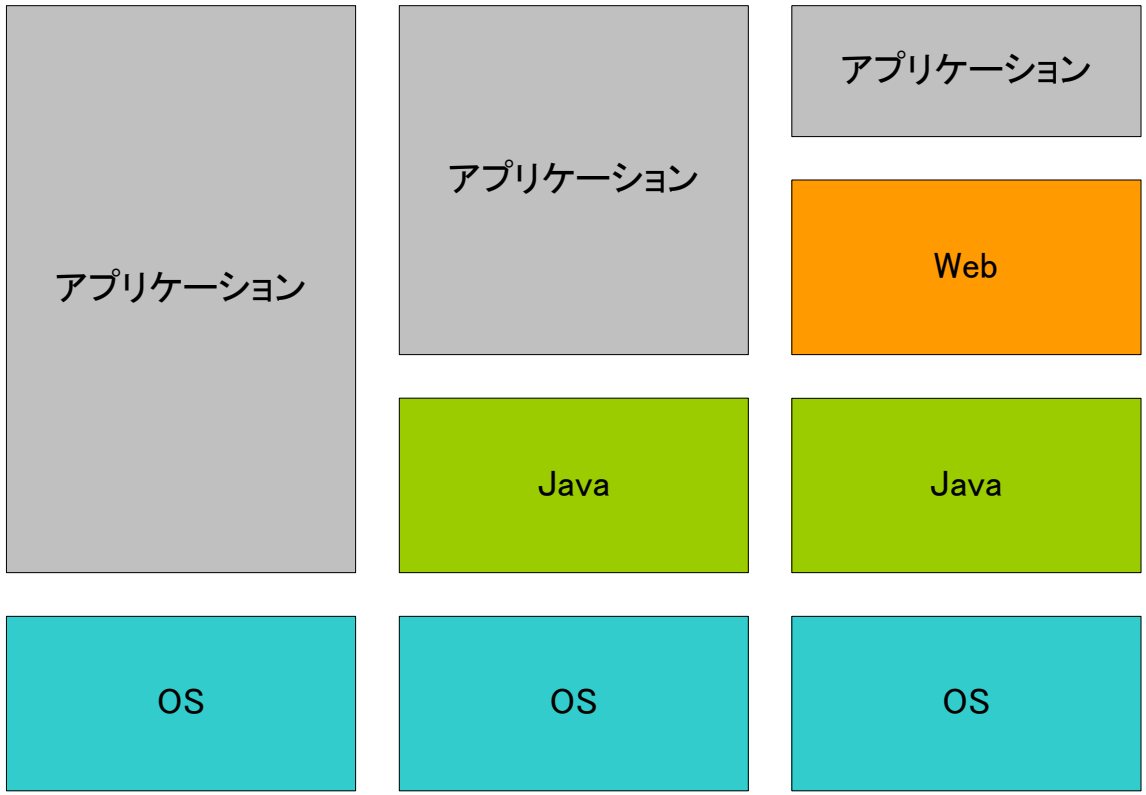
- クラウド時代を確認
- クラウド時代のアプリケーション開発
- edge2.ccの活動

クラウドとは

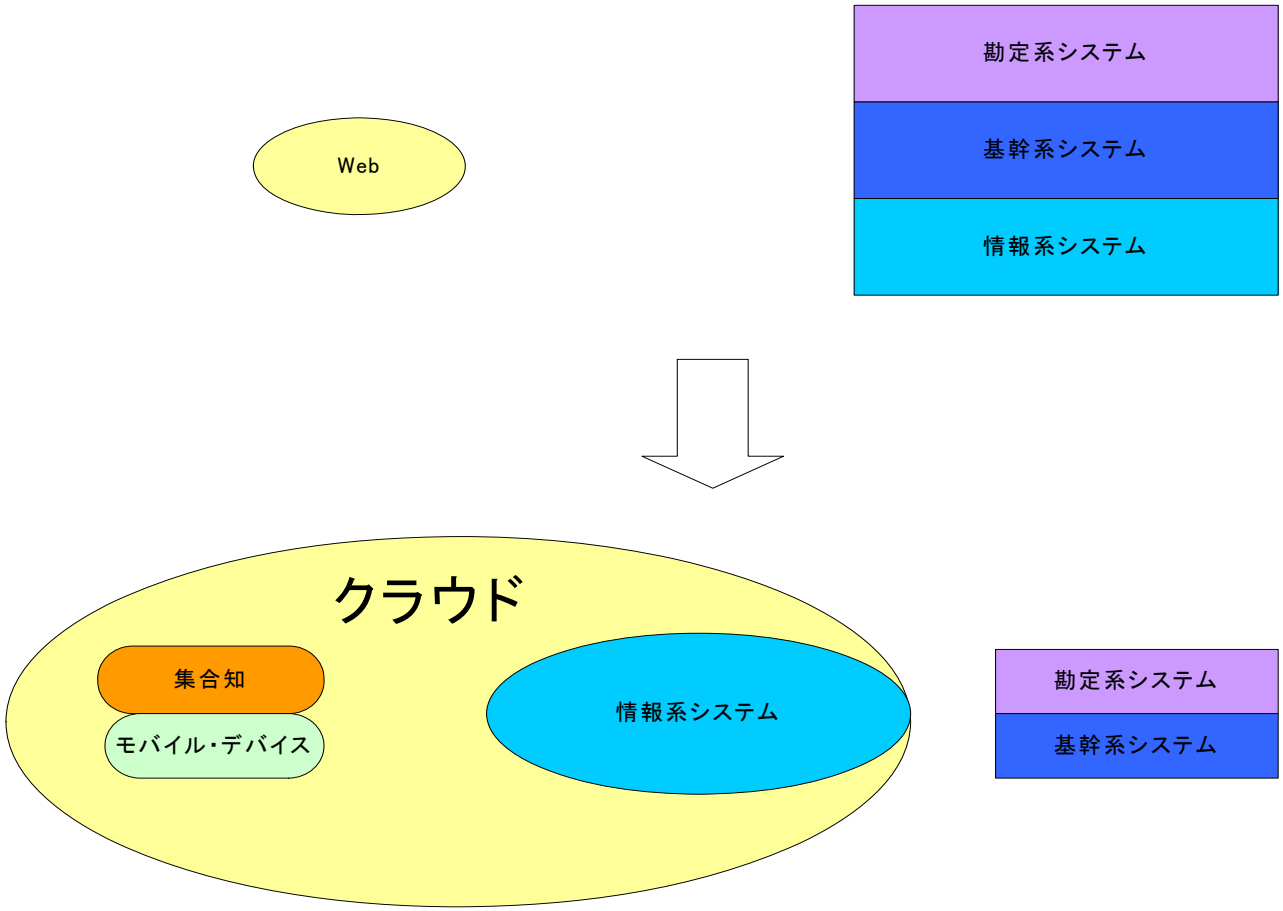


SaaS: Software as a Service
PaaS: Platform as a Service

Webがプラットフォームになる



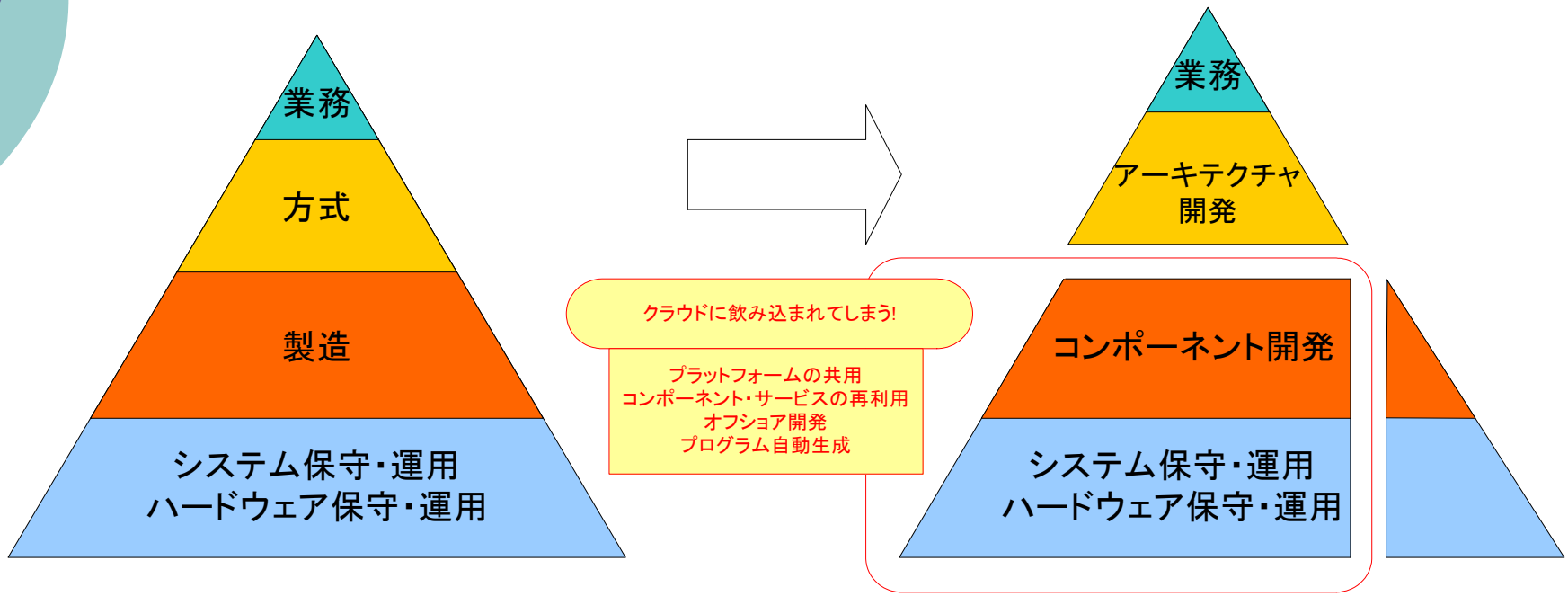
マーケットの変化


Web
勘定系システム
基幹系システム
情報系システム
クラウド
集合知
モバイル・デバイス
情報系システム
勘定系システム
基幹系システム

クラウド・コンピューティングの意義

- 「チープ革命」(Web 2.0)の実現
  - ソフトウェア開発・運用のコスト構造が激変
- クラウド時代にはさらに…
  - DSL駆動開発
    - プログラムの自動生成
  - オフショア開発
    - 単純開発は国内に残らない
  - CBD (Component-Based Development)
    - Webプラットフォーム上でのサービス・コンポーネントの再利用
    - マッシュアップ

クラウド時代のソフトウェア開発



DSL駆動開発＆コンポーネント

目次

- クラウド時代を確認
- クラウド時代のアプリケーション開発
- edge2.ccの活動

クラウド時代のモデリング

- CIM (Computer Independent Model)
  - 概念モデル
  - 変化なし
- PIM (Platform Independent Model)
  - 論理モデル
  - 少し変化：非同期、並列、分散への対応
- PSM (Platform Specific Model)
  - 物理モデル
  - 大きく変化：クラウド・プラットフォーム

アプリケーション統合の障壁

- Networks are unreliable.
- Networks are slow.
- Any two applications are different.
- Change is inevitable.

『Enterprise Integration Patterns』より

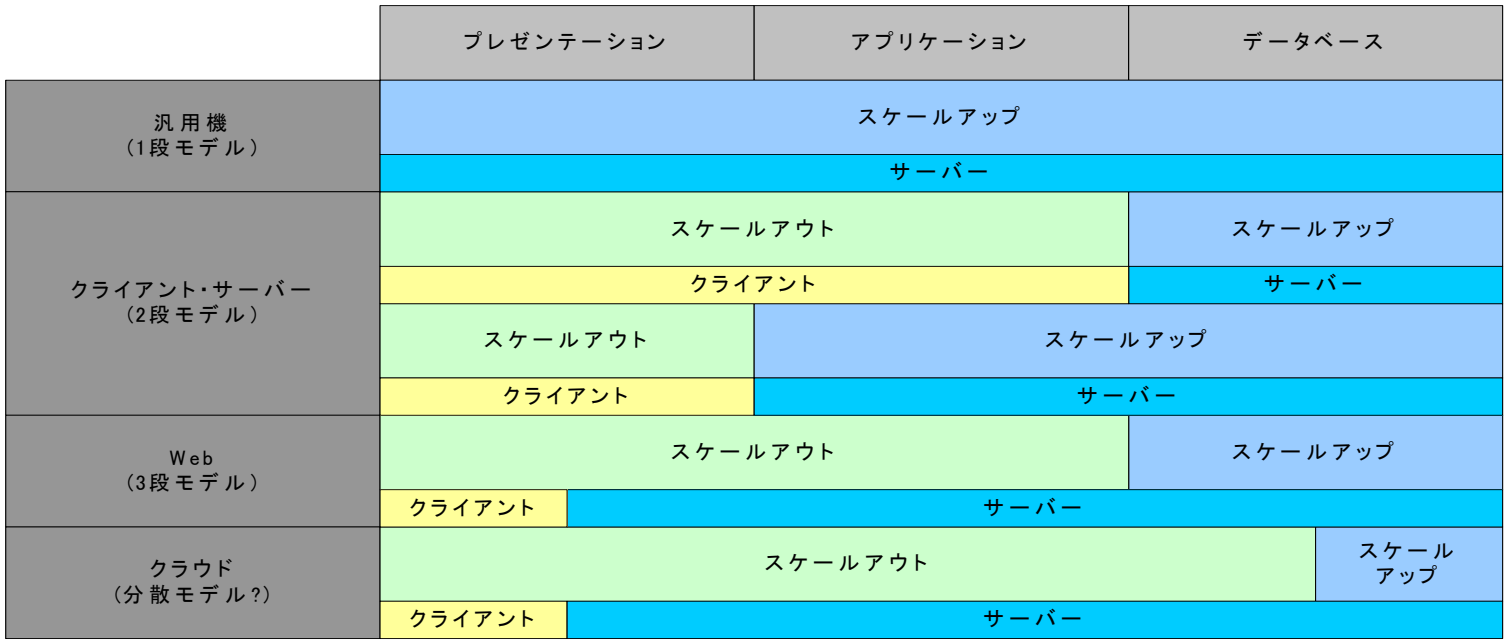
分散アプリケーションの作法

- エラーの発生を前提とする。
  - Networks is are unreliable.
- 入出力は非同期処理を前提とする。
  - Networks are slow.
- Webプラットフォームを前提とする。
  - Any two applications are different.
- アジャイル開発を前提とする。
  - Chenge is inevitable.

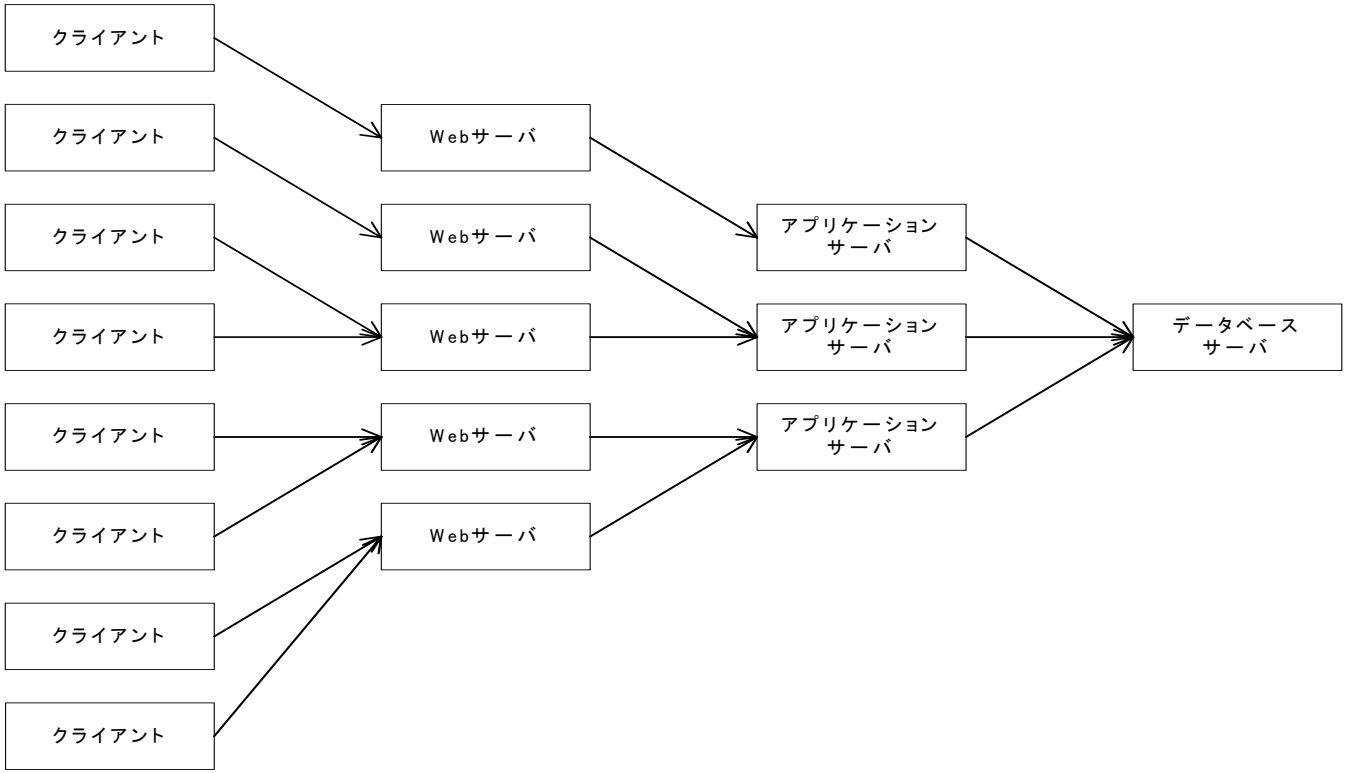
クラウド・アプリケーション

- Webアプリケーション
  - MVC2からMVCへ
  - クライアント/サーバ時代のGUIをWebで実現
  - HTML5
- スケーラビリティ
  - 非同期処理、並行処理、分散処理
  - ACIDからBASEへ
    - Key/Valueストレージ
- 分散アプリケーション
  - 故障と遅延への対応
  - 逐次処理から並行処理へ

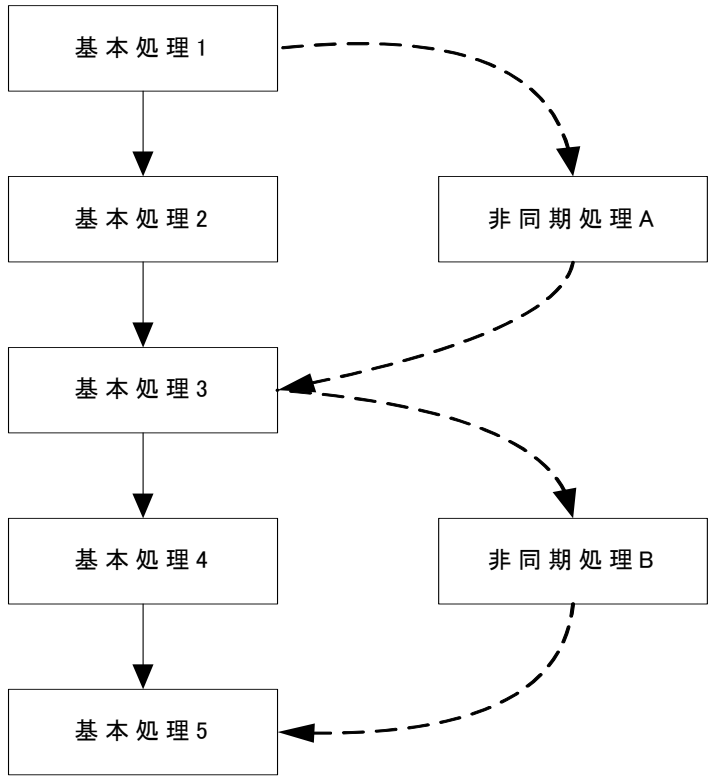
スケールアップとスケールアウト



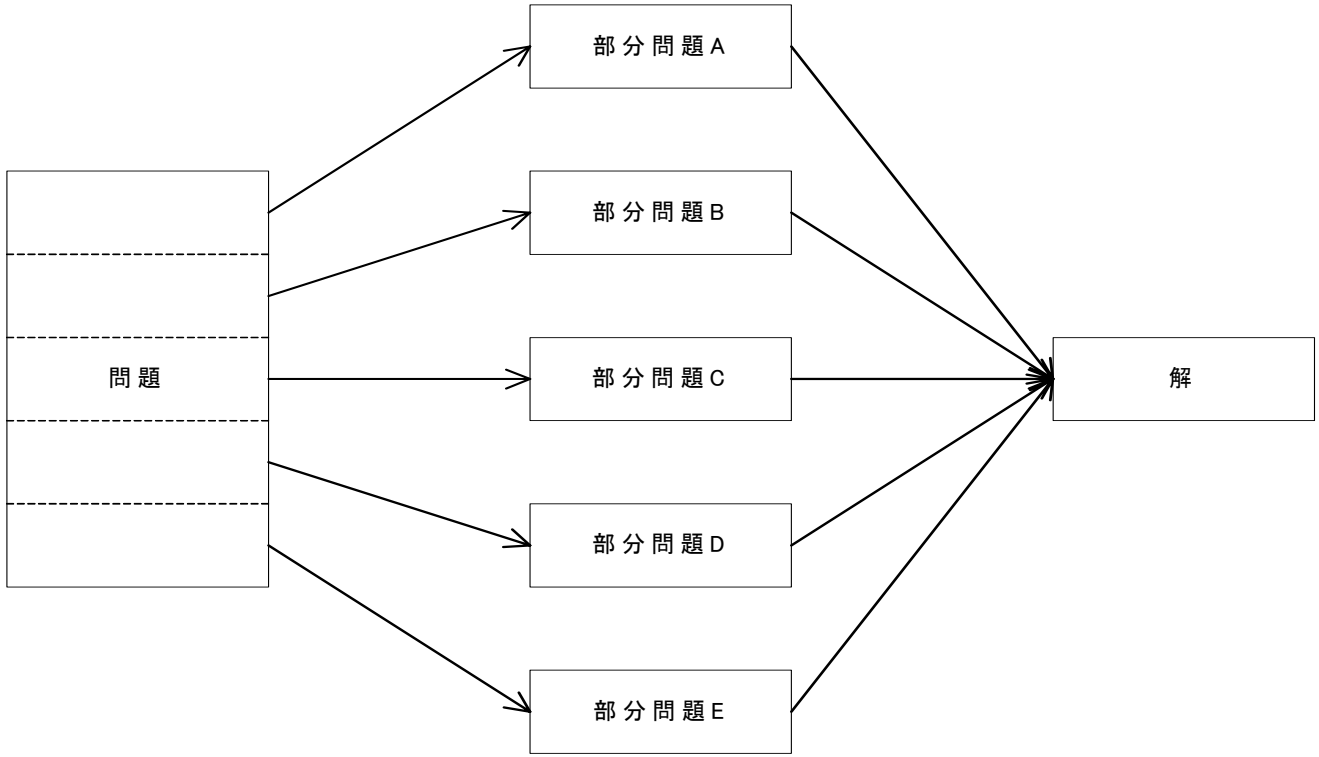
負荷分散


クライアント
クライアント
クライアント
クライアント
クライアント
クライアント
クライアント
クライアント
Webサーバ
Webサーバ
Webサーバ
Webサーバ
Webサーバ
アプリケーション
サーバ
アプリケーション
サーバ
アプリケーション
サーバ
データベース
サーバ

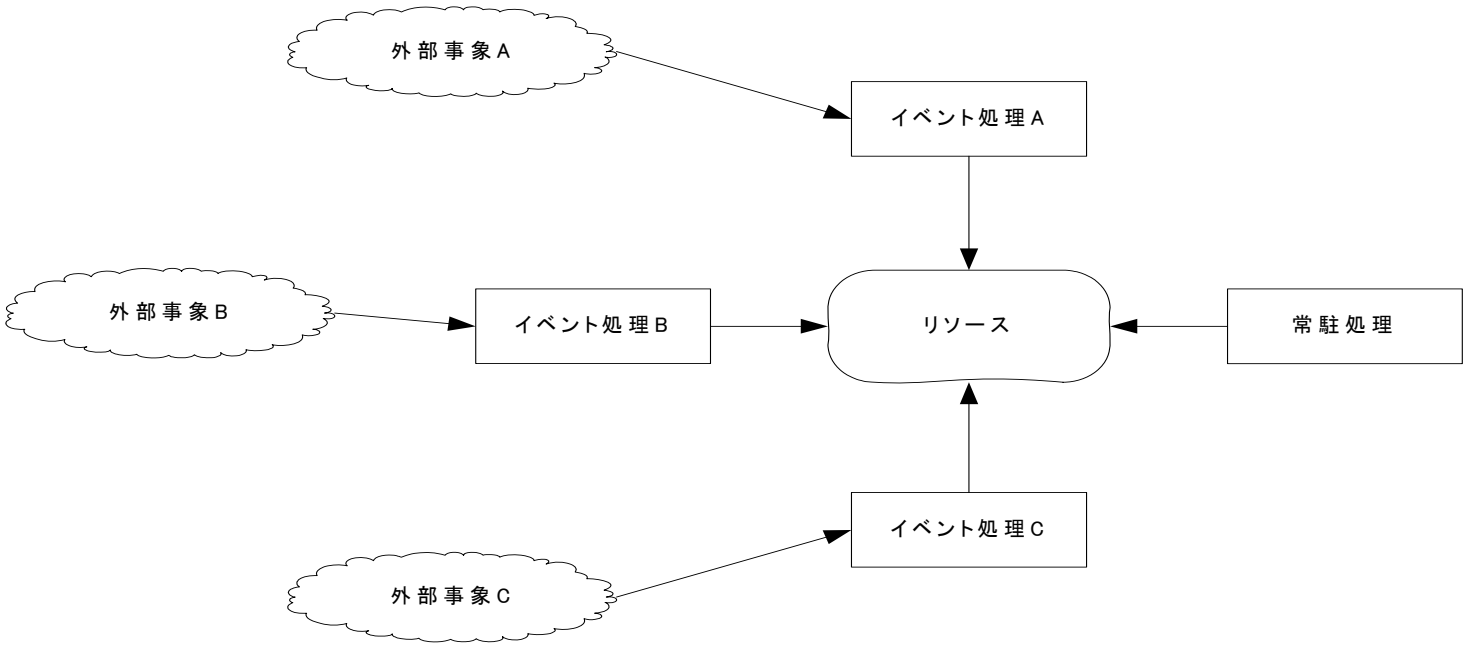
非同期処理



分散処理



イベント駆動処理



ACID vs BASE

- ACID
  - Atomicity、Consistency、Isolation、Durability
  - 高くても遅くても安全に
- BASE
  - Basically available、Soft state、Eventually consistent
  - 多少おかしくてもよいので安く早く

データの種別

- スコープ
  - セッション
  - サービス
  - ビジネス・プロセス
  - エンタープライズ
  - インターネット
- アイソレーション・レベル
  - READ UNCOMMITTED
  - READ COMMITTED
    - Dirty Readを防ぐ
  - REPEATABLE READ
    - Non-repeatable Readを防ぐ
  - SERIALIZABLE
    - Phantom Readを防ぐ
- 更新頻度
  - read-only
  - read-mostly
  - read-write

データの性質によって戦略が異なる

- 銀行内の口座振替
  - スコープ：エンタープライズ
  - アイソレーション・レベル：SERIALIZABLE
  - 更新頻度：read-write

  ⇒RDBMS、スケールアップ
- サーチエンジンの途中結果
  - スコープ：セッション
  - アイソレーション・レベル：READ UNCOMMITTED
  - 更新頻度：read-only

  ⇒クラウドDB、スケールアウト

エンティティ・モデルとデータ種別

- SimpleModelingの場合、エンティティの種別をステレオタイプで指定する
  - ステレオタイプの組合せでアクセス・パターンを絞り込む
- たとえば…
  - Actor
    - read-mostly, BASE
  - Event
    - append-only, BASE
  - Resource
    - read-write, ACID
  - Powertype
    - read-only, マスターデータとして配布(分散キャッシュ)

工夫すべきこと

- BASEで十分な応用を見つける
- 利用者が不整合を許容する隙間(inconsistent window)を見つける
- UI上の工夫で利用者に気付かれないようにする

エンティティ・モデルとユースケース

- SimpleModelingはユースケース駆動
  - 業務ユースケース、システム・ユースケース
- ユースケースのフロー(物語の脚本)に参加するエンティティの挙動を分析
  - アクセス・パターンを絞り込む
  - 更新頻度
  - 要求されるレスポンスタイム
  - 使用するアクターやサービスの局所性

クラウド・アプリケーションの三つの技術

- UI
  - MVC2からMVCへ
- データベース
  - KVS over RDBMS
- 通信方式
  - メッセージング指向

UI：MVC2からMVCへ

- HTML5(＋Open Web Platform)によって、Webブラウザ上で通常の(クライアント・サーバ時代の)GUIの構築が可能になる。
  - Web GUI
- MVC2からMVCへ
  - MVC2はWebアプリケーション向けのアーキテクチャ・パターンで本来のMVCとは異なる。
  - WebアプリケーションむけのMVC2から、GUIアプリケーション向けのMVCへ

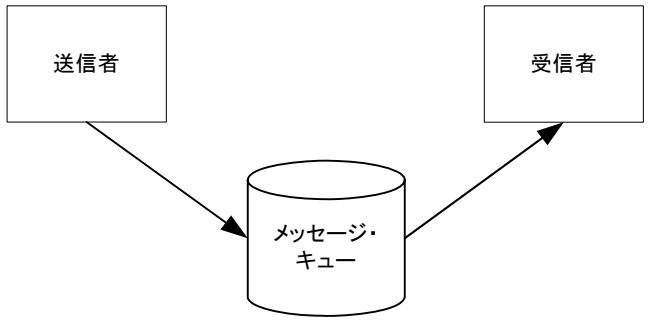
データベース：KVS over RDBMS

- KVS(Key/Value Store)
  - クラウドで用いられる大規模分散データストア
  - Key/Value型
- KVSの用途
  - スケーラビリティの必要な大容量データ
  - 超安価なデータストア
- KVSはRDBMSの以下の機能を持っていない。
  - トランザクション処理
  - SQLのJOIN、集計機能など
  - ER図でのモデリングが(そのままでは)利用できない
- KVSとRDBMSの使い分けが重要なポイント
  - できるだけKVSを使いたい

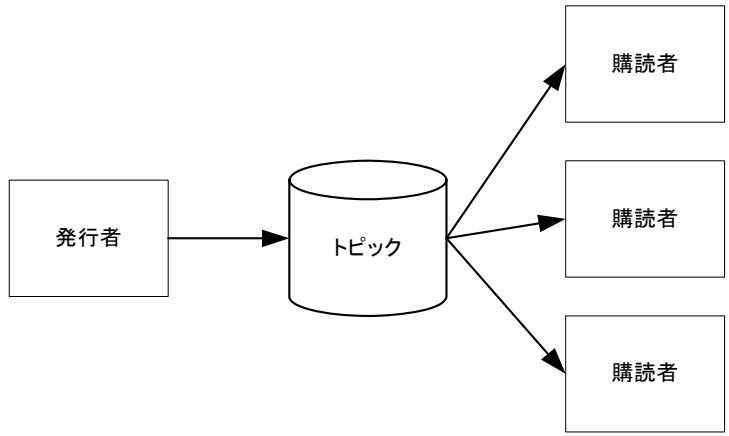
通信基盤:メッセージング指向

- 非同期、並列、分散を処理するために、メッセージング機能を活用。
- MOM(Message Oriented Middleware)
  - Peer-to-Peer, Publisher/Subscribe
  - Javaの場合はJMS(Java Messaging Service)
- Google App Engineではまだ用意されていない
  - TaskQueue機能を用いてアプリケーションで実装する必要がある。
- メッセージングを行うための統合フレームワーク
  - メッセージングの次の段階
  - Apache Camelなど

MOMの種類

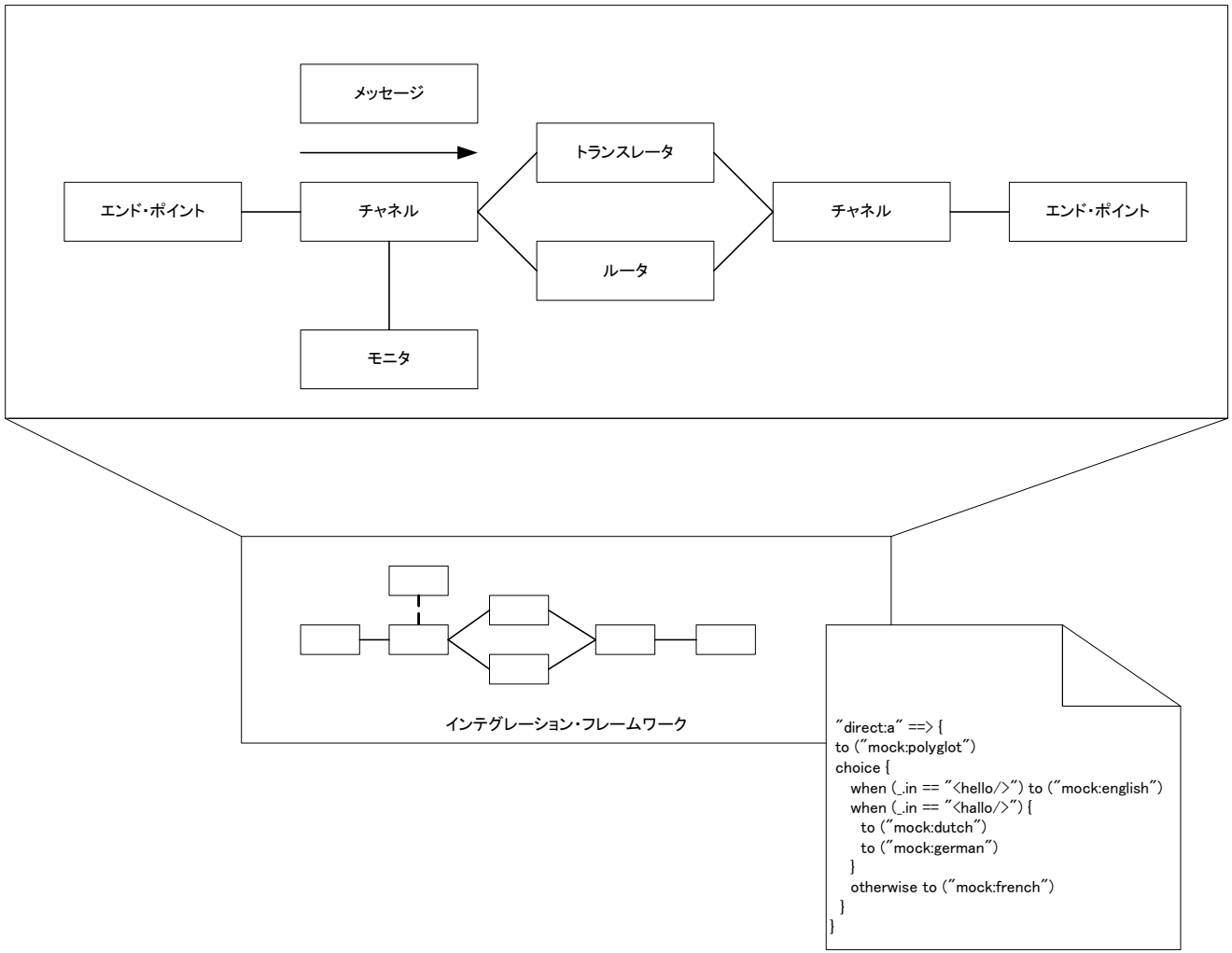


Peer-To-Peer

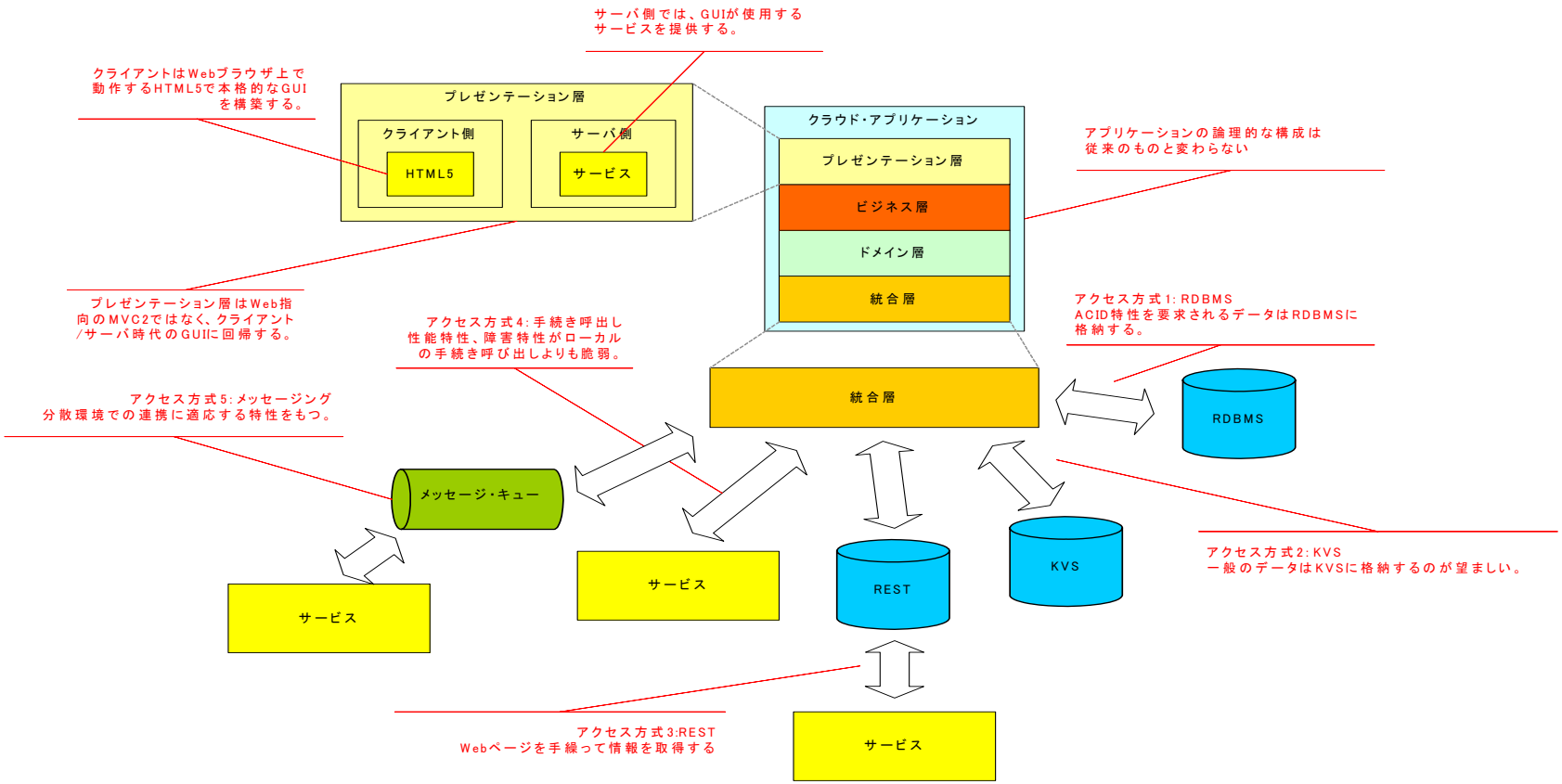


Publish/Subscribe

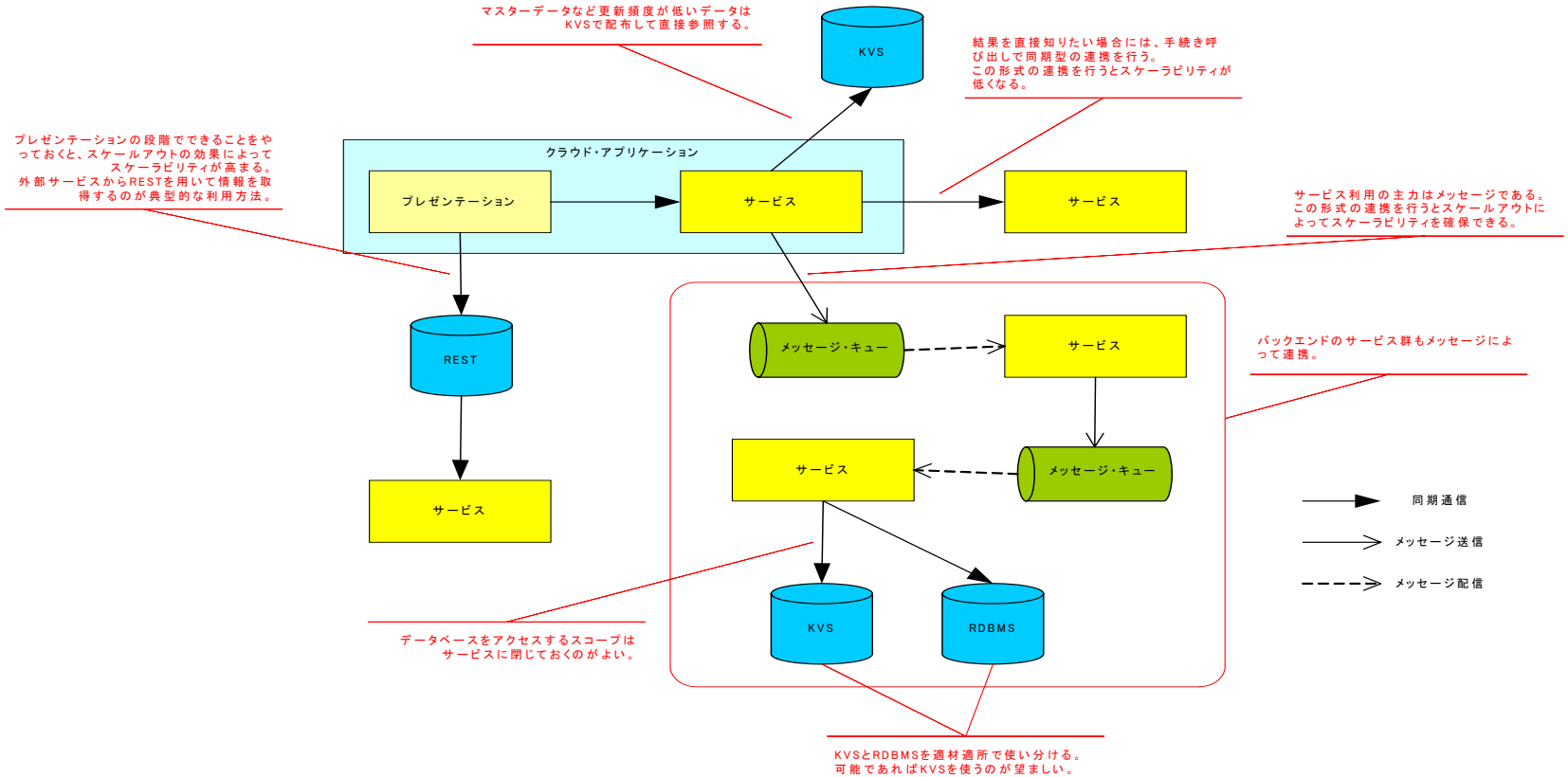
統合フレームワーク


メッセージ
エンド・ポイント
チャネル
トランスレータ
ルータ
チャネル
エンド・ポイント
モニタ
インテグレーション・フレームワーク
"direct:a" ==> {
to ("mock:polyglot")
choice {
when (_in == "<hello/>") to ("mock:english")
when (_in == "<hallo/>") {
to ("mock:dutch")
to ("mock:german")
}
otherwise to ("mock:french")
}
}

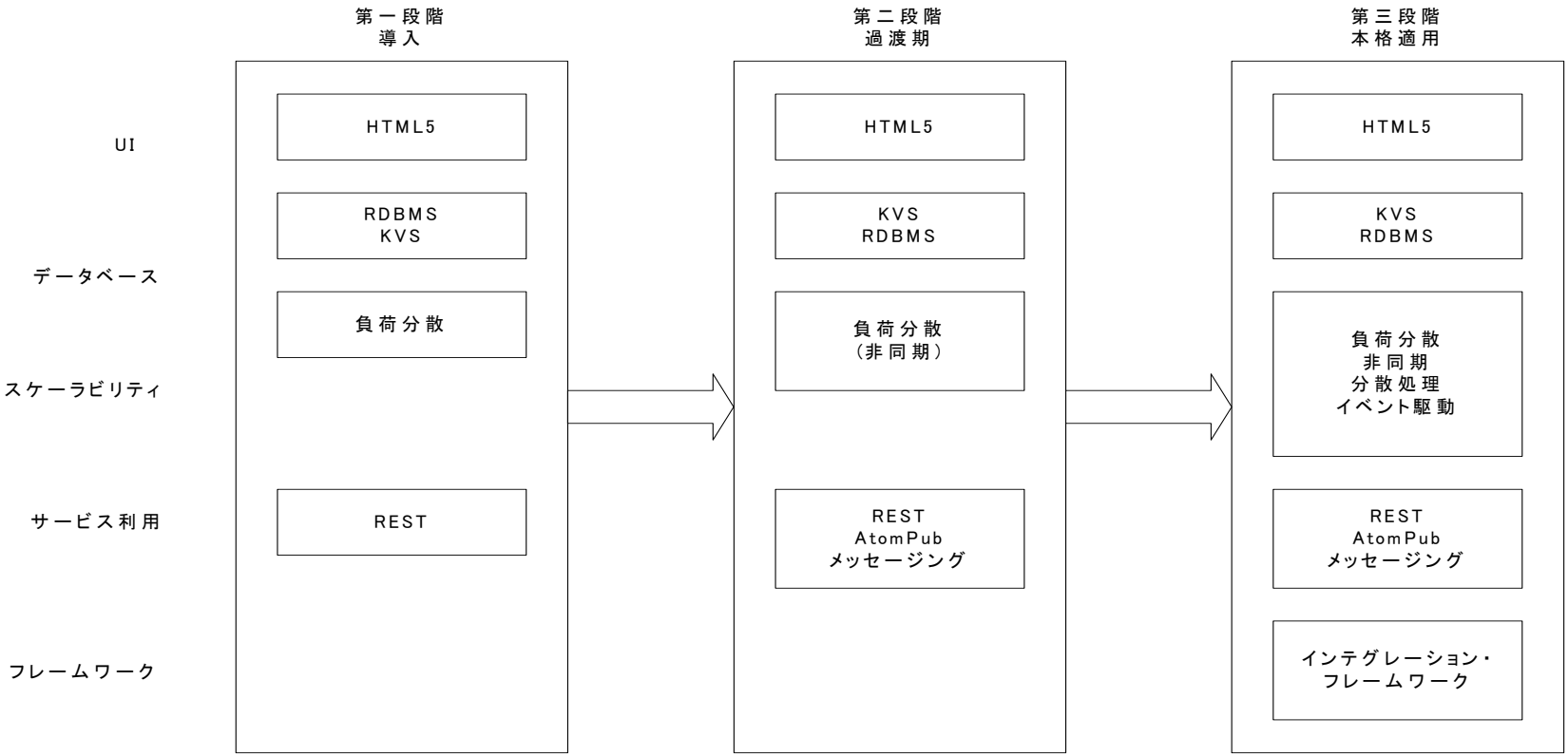
クラウド・アプリケーションのアーキテクチャ



クラウド・アプリケーションのアーキテクチャ例



クラウド・アプリケーションへの移行パス



クラウド・アプリケーションの設計技法

- 概念モデル(要求モデル)は今までどおり
  - ドメイン・モデル
  - ユースケース・モデル
- 論理モデル(PIM, Platform Independent Model)
  - 非同期、並列、分散を本格的に取り込む
  - メッセージングによる分散コンポーネントの非同期通信
- 物理モデル(PSM, Platform Specific Model)
  - データ・モデルのKVS化
    - 非正規化(データ集約)、データ分割
  - Web技術に対応
    - HTML5, AtomPub
  - 分散技術に対応
    - メッセージング
  - メッセージングを軸とした統合フレームワークの導入

目次

- クラウド時代を確認
- クラウド時代のアプリケーション開発
- edge2.ccの活動

edge2.cc

- Edge to Cloud Computing
- http://www.edge2.cc
- モデル駆動開発×クラウドコンピューティングの実証プロジェクト
- アプリケーション開発者の立場から、クラウド・アプリケーションの開発技法を確立する
  - 要素技術、アーキテクチャ、モデリング、モデル駆動開発
  - モデル駆動開発の技術として SimpleModeling&SimpleModelerを採用

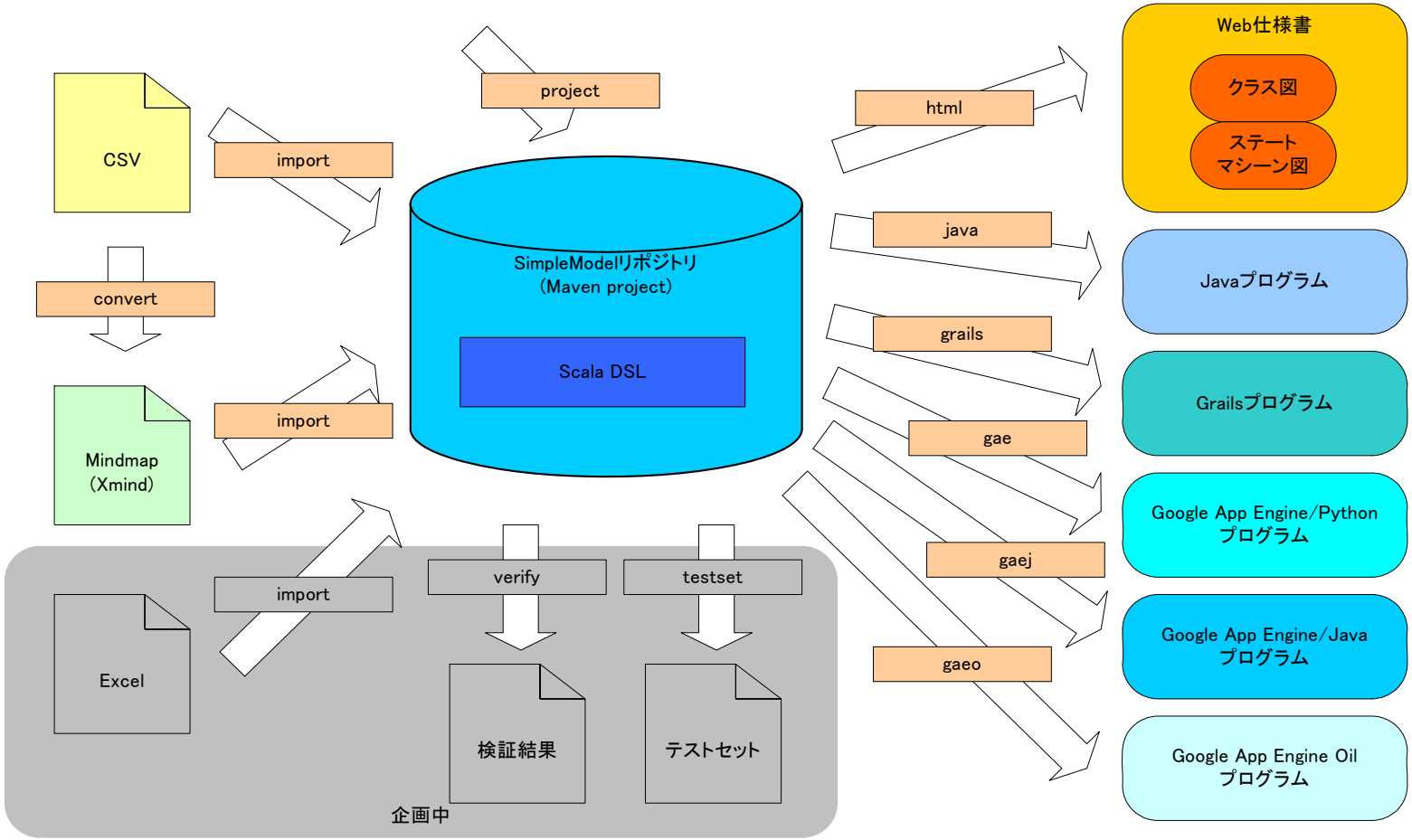
SimpleModeler

- SimpleModel用モデル・コンパイラ
- テキストDSL
  - Scala DSL
  - Scala DSL＆mavenによるモデル・リポジトリ
- Web仕様書、Java、Grails、Google App Engine Python/Javaを生成

Scalaについて

- 純粋オブジェクト指向言語＋本格関数型言語
  - 関数型言語とクラウドは相性がよいはず
- 静的型付
- Java VM上で動作
- 言語仕様が複雑
  - その代わり浅海の体感ではJavaの3倍ぐらいの生産性がある
- DSLのホスト言語として充実した機能を持っている
- モデル・コンパイラの実装言語として充実した機能を持っている

SimpleModelerの動作



SimpleModeler
CSVで記述できること

yorozu.csv

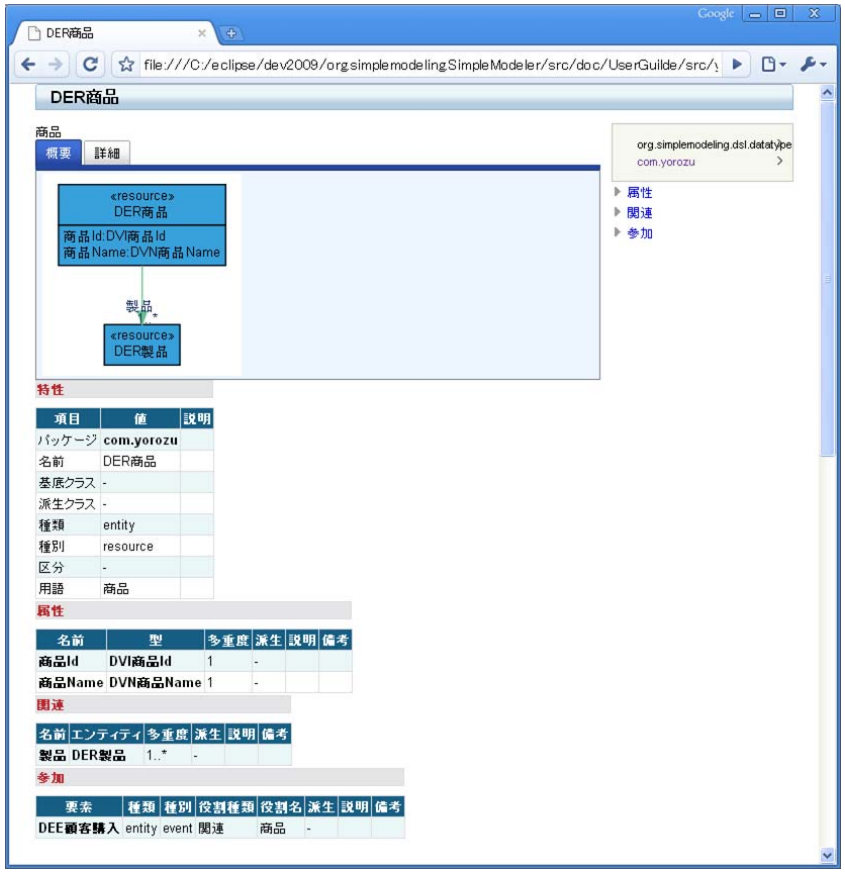
```
#actor,base,parts,attrs,powers,states,roles
顧客,,,住所
個人顧客,顧客,,,性別(男性;女性)
法人顧客,顧客
従業員,,,,,,店員
#role
店員
#resource
商品,,製品+,,,商品状態(入荷待;在庫中;配送中;販売完)
製品
#event
顧客取引,,顧客;店員
顧客購入,顧客取引,商品+
```

SimpleModeler
マインドマップ(XMind)

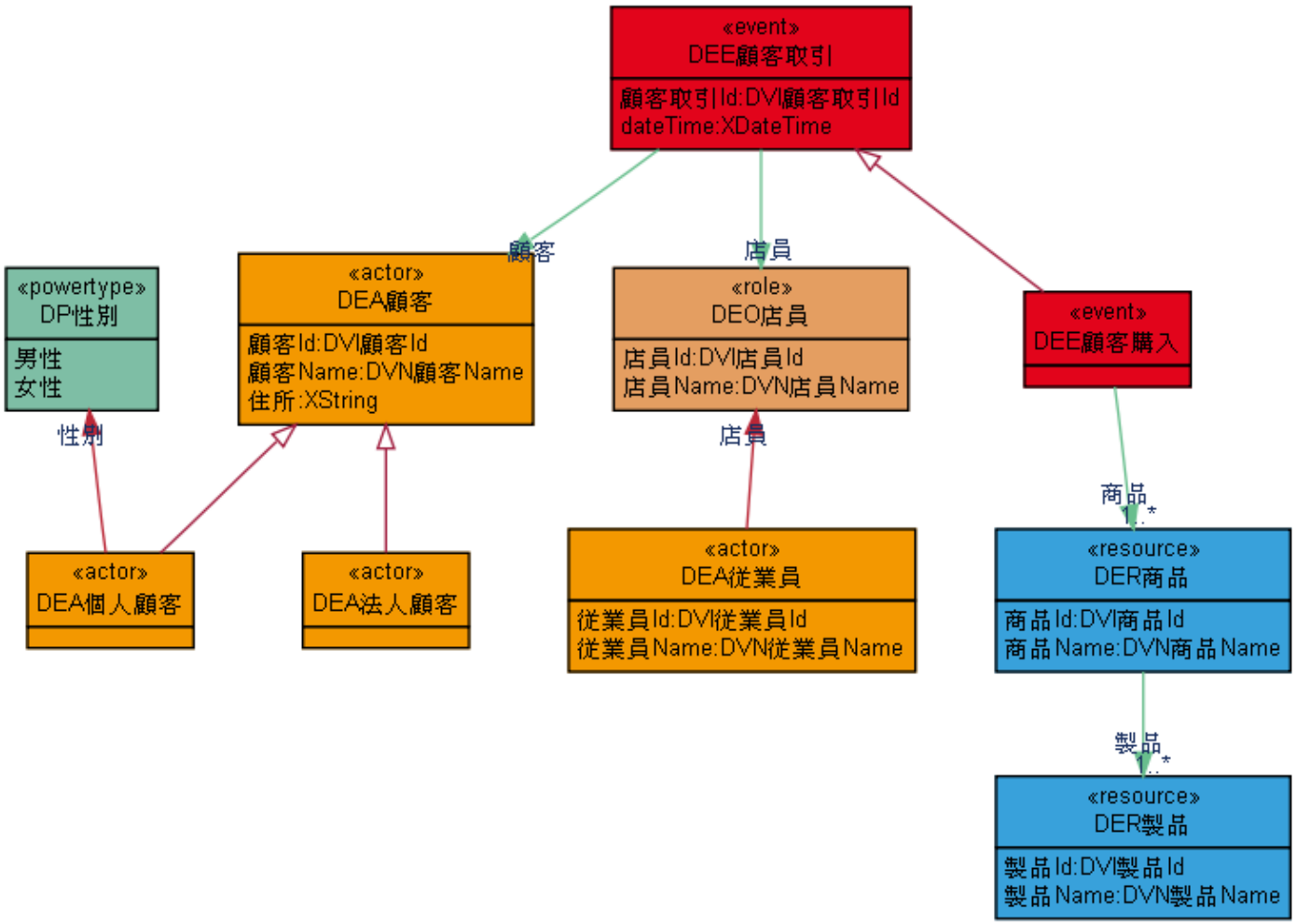
SimpleModeler
Scala DSL

```
package com.yorozu

import org.simplemodeling.dsl._
import org.simplemodeling.dsl.datatype._
import org.simplemodeling.dsl.domain._
import org.simplemodeling.dsl.domain.values._

case class DER製品 extends DomainResource {
  term = "製品"
  caption = "製品"
  brief = <t></t>
  description = <text></text>

  id("製品Id", DVI製品Id())
  attribute("製品Name", DVN製品Name())
}

case class DVI製品Id extends DomainValueId {
  term = "製品Id"
  caption = "製品Id"
  brief = <t></t>
  description = <text></text>

  attribute("value", XString)
}

case class DVN製品Name extends DomainValueName {
  term = "製品Name"
  caption = "製品Name"
  brief = <t></t>
  description = <text></text>

  attribute("value", XString)
}
```

SimpleModeler
Web仕様書



SimpleModeler
クラス図

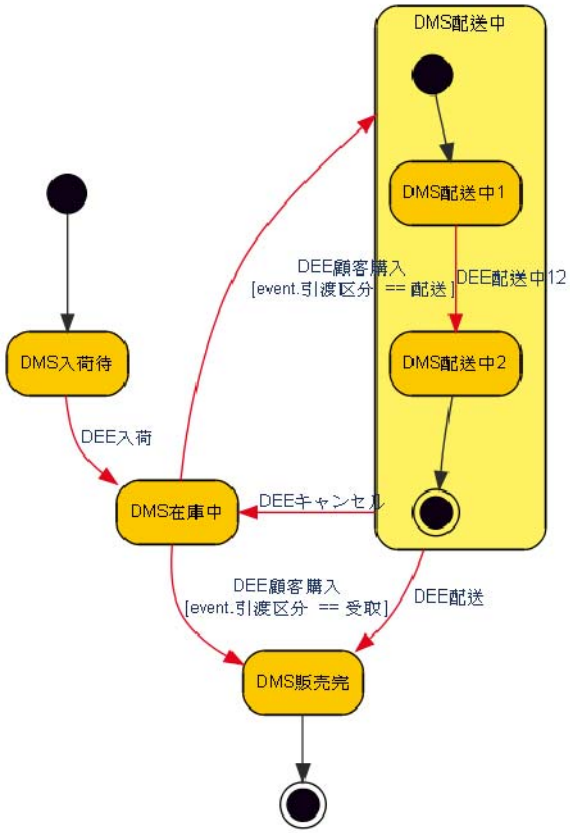


SimpleModeler
Scala DSL→ステートマシーン図

```
package com.yorozu

import org.simplemodeling.dsl._
import org.simplemodeling.dsl.datatype._
import org.simplemodeling.dsl.domain._
import org.simplemodeling.dsl.domain.values._

case class DER商品 extends DomainResource {
  term = "商品"
  caption = "商品"
  brief = <t></t>
  description = <text></text>

  id("商品Id", DVI商品Id())
  attribute("商品Name", DVN商品Name())
  association("製品", DER製品(), OneMore)
  statemachine(DM商品状態())
}

・・・中略・・・
case class DM商品状態 extends DomainStateMachine {
  term = "商品状態"
  caption = "商品状態"
  brief = <t></t>
  description = <text></text>

  state(DMS入荷待())
  state(DMS在庫中())
  state(DMS配送中())
  state(DMS販売完())
}
```

```
case class DMS入荷待 extends DomainState {
  term = "入荷待"
  caption = "入荷待"
  brief = <t></t>
  description = <text></text>

  transition(DEE商品入荷(), DMS在庫中())
}

case class DMS在庫中 extends DomainState {
  term = "在庫中"
  caption = "在庫中"
  brief = <t></t>
  description = <text></text>

  transition(DEE顧客購入(), DMS配送中())
  transition(DEE顧客購入(), DMS販売完())
}

case class DMS配送中 extends DomainState {
  term = "配送中"
  caption = "配送中"
  brief = <t></t>
  description = <text></text>

  transition(DEE商品配送(), DMS販売完())
}

case class DMS販売完 extends DomainState {
  term = "販売完"
  caption = "販売完"
  brief = <t></t>
  description = <text></text>

}
```

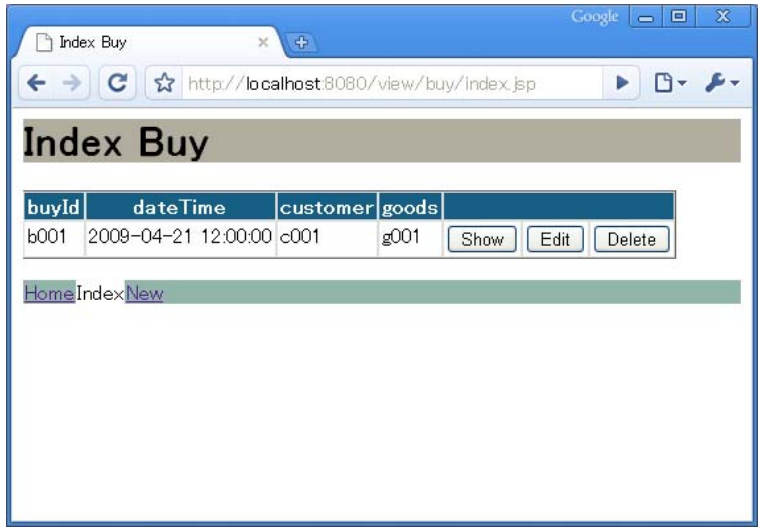
SimpleModeler
ステートマシーン図と状態遷移表

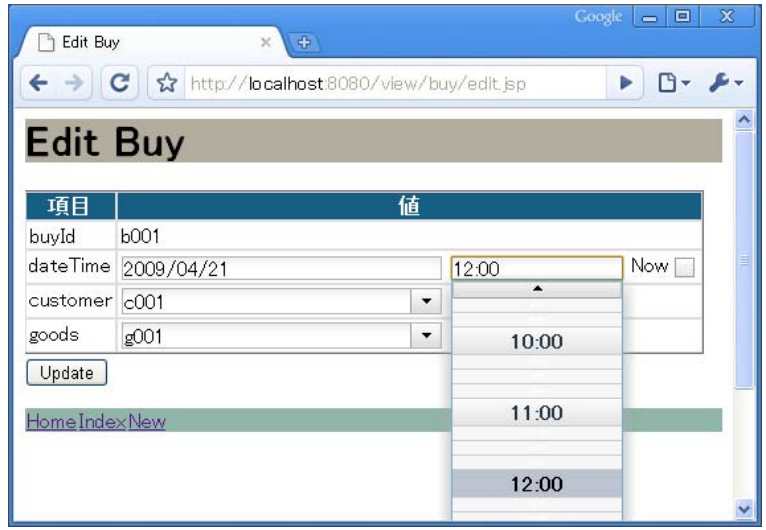


		自動	DEE入荷	DEE顧客購入		DEE配送	DEEキャンセル	DEE配送中12
				event.引渡区分 == 配送	event.引渡区分 == 受取			
初期状態		DMS入荷待	N/A	N/A	N/A	N/A	N/A	N/A
DMS入荷待		N/A	DMS在庫中	N/A	N/A	N/A	N/A	N/A
DMS在庫中		N/A	N/A	DMS配送中	DMS販売完	N/A	N/A	N/A
DMS配送中	DMS配送中	N/A	N/A	N/A	N/A	DMS販売完	DMS在庫中	N/A
	初期状態	DMS配送中1	N/A	N/A	N/A	N/A	N/A	N/A
	DMS配送中1	N/A	N/A	N/A	N/A	N/A	N/A	DMS配送中2
	DMS配送中2	終了状態	N/A	N/A	N/A	N/A	N/A	N/A
	終了状態	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DMS販売完		終了状態	N/A	N/A	N/A	N/A	N/A	N/A
終了状態		N/A	N/A	N/A	N/A	N/A	N/A	N/A

SimpleModeler

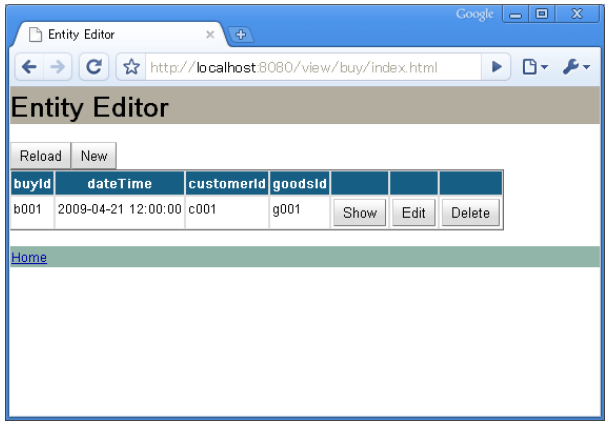
Google App Engine/Java Servlet/JSP/Dojo Toolkit

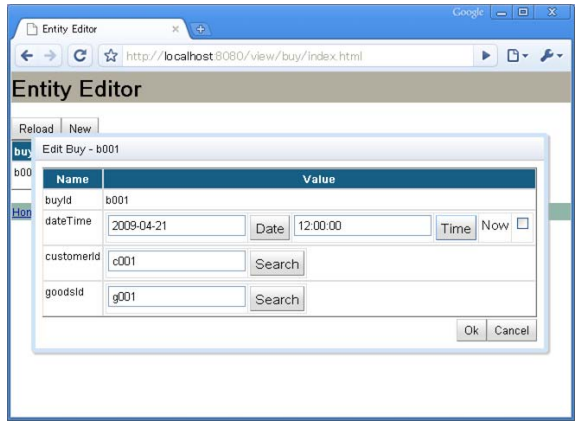


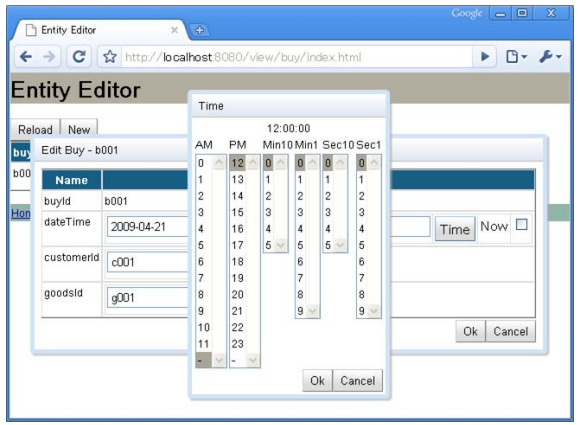


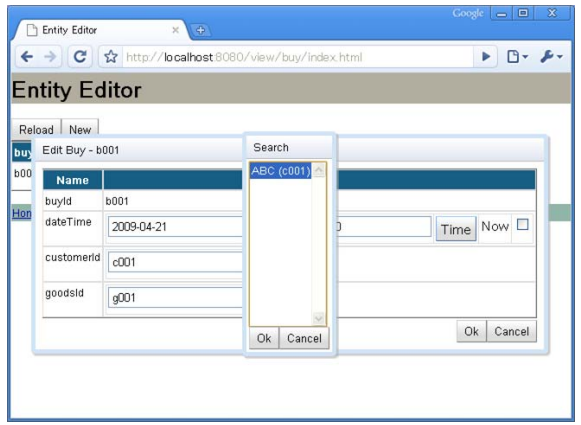
SimpleModeler

Google App Engine/Java
Google Web Toolkit

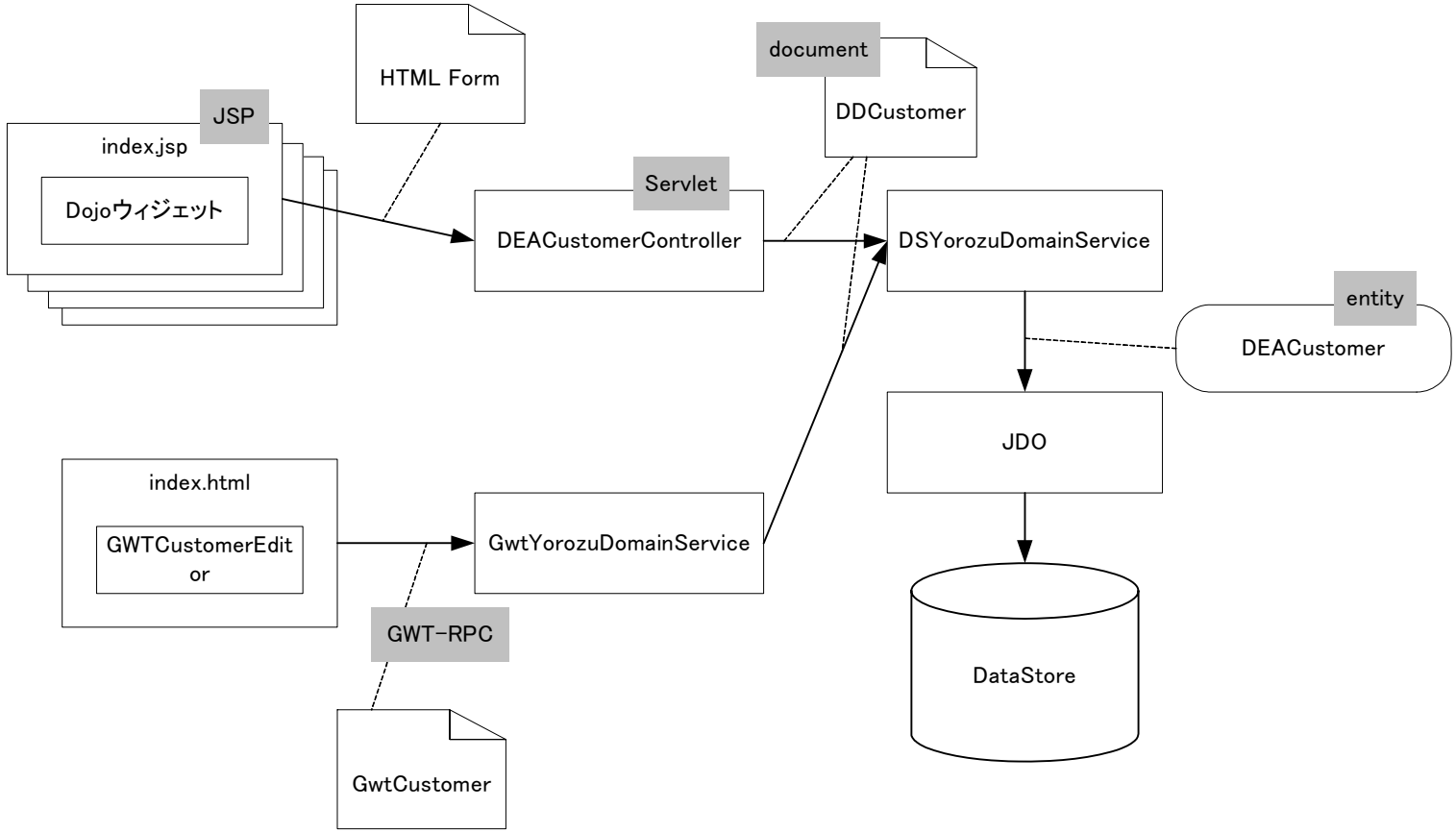








Google App Engine/Java アプリケーション構成



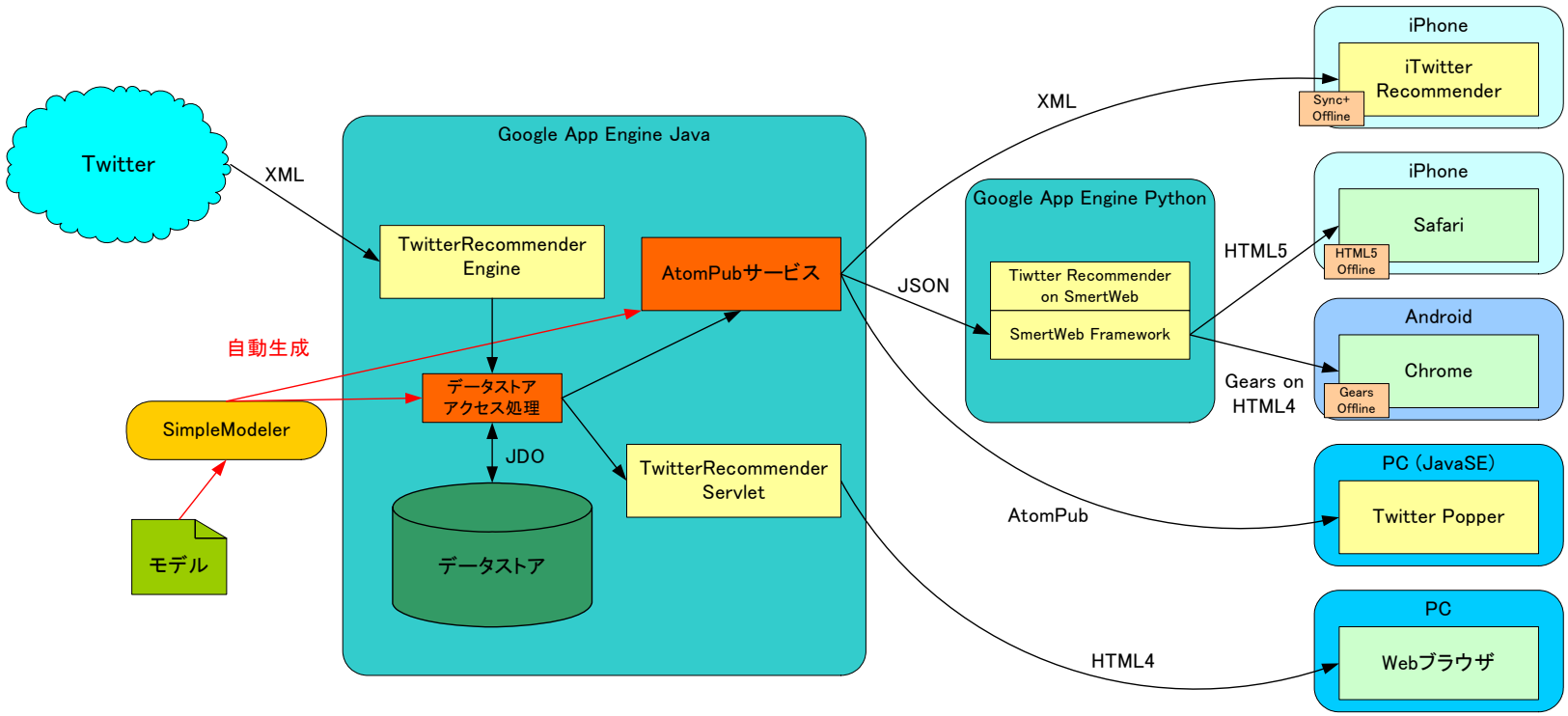
SimpleModelerによるDSL駆動アプローチ

- ドメイン・モデル(概念モデル、論理モデル)からクラウド向けの物理モデルの自動生成
  - クラウド・アプリケーション開発で難易度が高く、煩雑なプログラムを自動生成
- ユースケース・モデル
  - ドメイン・モデルの正当性を検証
  - サービス・モデルを抽出
- サービス・モデル
  - RESTやAtomPubなどのサービスのAPIとエントリポイントを自動生成
- edgeSNSの開発を通じて実用化していく

TwitterRecommender

- Twitterを使用した集合知アプリケーション
  - Twitterから収集したフレンド、フォロワーのリンクからソーシャルグラフを生成して、フォロワーの推奨を行う
  - 収集した情報をPC, iPhone, Androidで表示
- 目的
  - Google App Engine/Python, Javaの味見
  - 集合知アプリケーションの味見
  - SimpleModelerの活用(DSL駆動開発)
  - モバイル技術
- Google Developer Day Japan 2009のSandboxに出展

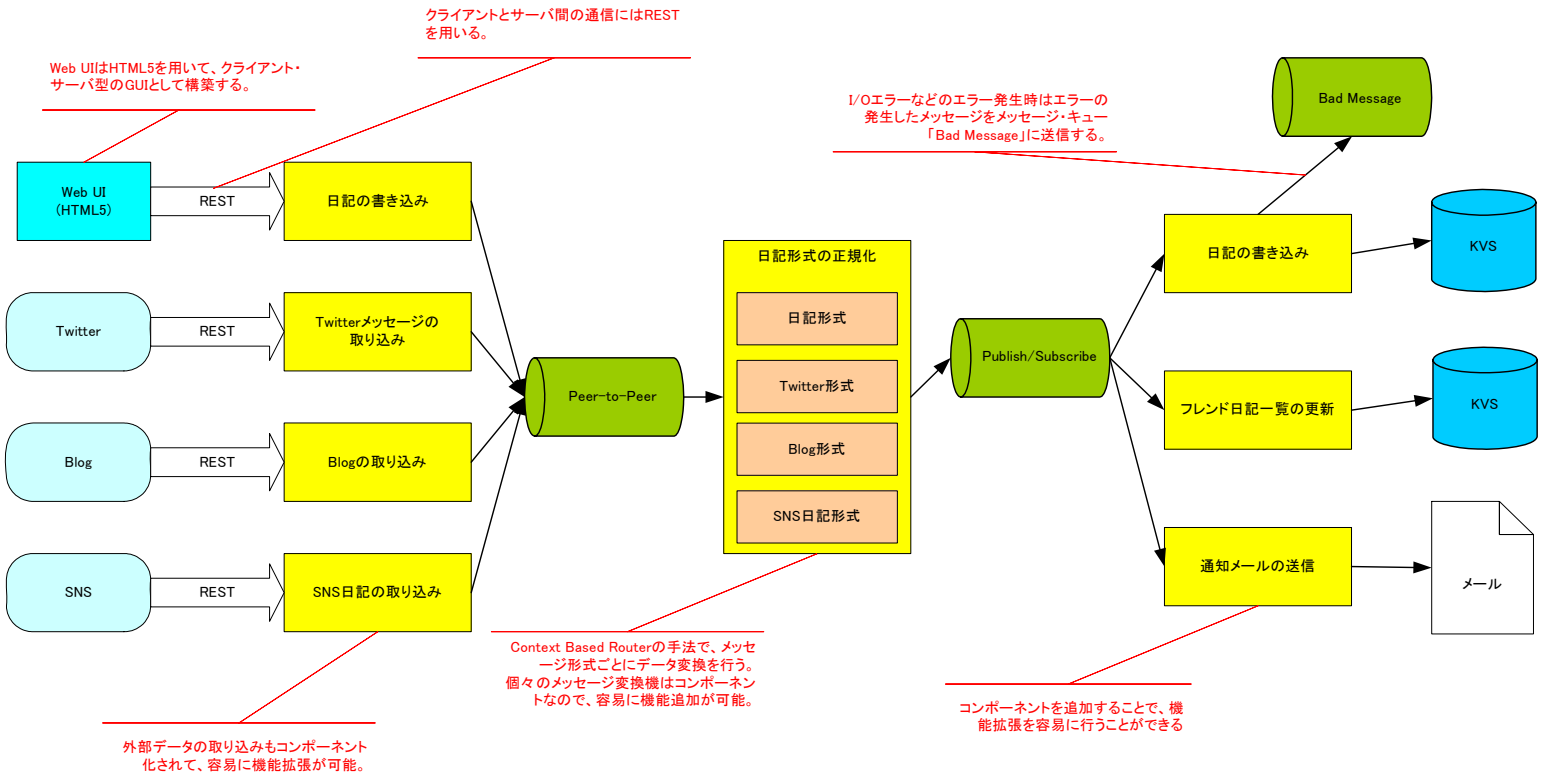
TwitterRecommender



edgeSNS

- 簡易版SNS
  - SNS日記機能の実現
  - クラウド・アプリケーションの構築技術を追求するのを目的に、アプリケーションは平凡なものを選択
- 目的
  - メッセージングの活用
  - 非同期入出力の活用
  - メッセージング、非同期入出力の実現に対するSimpleModelerの活用
  - メッセージングを基盤にしたコンポーネント・ベース開発

edgeSNS


クライアントとサーバ間の通信にはREST
を用いる。
Web UIはHTML5を用いて、クライアント・
サーバ型のGUIとして構築する。
Web UI
(HTML5)
REST
日記の書き込み
Twitter
REST
Twitterメッセージの
取り込み
Blog
REST
Blogの取り込み
SNS
REST
SNS日記の取り込み
外部データの取り込みもコンポーネント
化されて、容易に機能拡張が可能。
Peer-to-Peer
日記形式の正規化
日記形式
Twitter形式
Blog形式
SNS日記形式
Context Based Routerの手法で、メッセ
ージ形式ごとにデータ変換を行う。
個々のメッセージ変換機はコンポーネン
トなので、容易に機能追加が可能。
Publish/Subscribe
I/Oエラーなどのエラー発生時はエラーの
発生したメッセージをメッセージ・キュー
「Bad Message」に送信する。
Bad Message
日記の書き込み
KVS
フレンド日記一覧の更新
KVS
通知メールの送信
メール
コンポーネントを追加することで、機
能拡張を容易に行うことができる

まとめ

- クラウド時代に入ってオープンシステム、Java(OO&Web)と同等の新たな技術体系の転換が発生する
- クラウド時代のアプリケーションは、クラウド流のWebアプリケーション
  - クラウドを使わない場合でも、クラウド指向の技術を用いることになる
  - UI⇒HTML5
  - データベース⇒KVS
  - 通信基盤⇒メッセージング