



Romain Bernard - Extreme Modeling

David Brocard - Consultant Indépendant

<http://davidbrocard.org>

Présentation

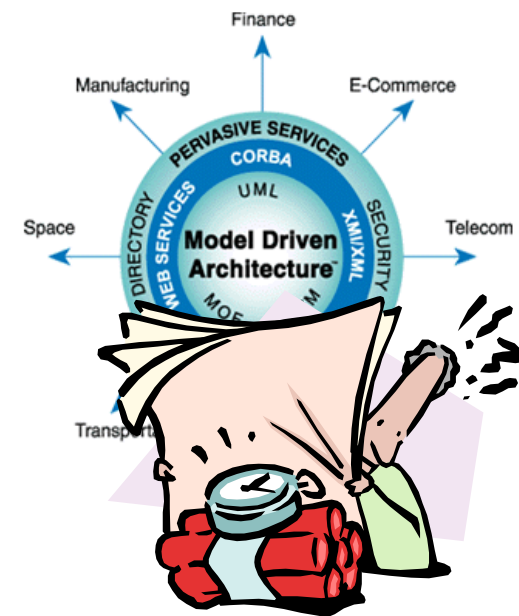
Model Driven Agile

Sommaire

- √ Les principes du Model Driven (MD) •
- √ Comment MDSD s'inscrit dans les concepts agiles
 - Parcours des valeurs, principes et pratiques
 - Rôles
 - Cycle de vie
- √ Conclusions

Les limites du "RUP®-MDA®-UML®"

- La plupart des projets MD sont de type RUP-MDA-UML :
 - § Outils trop généralistes
 - § Profils UML trop bas niveau
 - § Générateurs généralistes sur étagère...
 - § MDA inapplicable aujourd'hui (faute d'outils)•
 - § Les lourdeurs du RUP
 - § Quel intérêt à utiliser MDA® ?
 - § Trop grande généralité d'UML
 è grande complexité du langage
- Heureusement, il y a d'autres façons de pratiquer le MD



Principes de base du MD

I Des Modèles

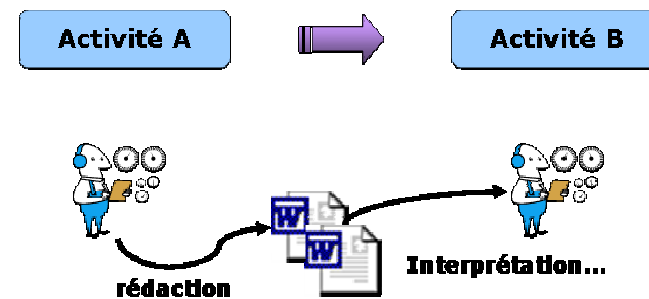
Langages formels et structurés

Métaphores adaptées à un problème

Adaptés aux processus cognitifs de résolution de problèmes

Exploitable automatiquement par des « composants MD » (transformateurs, générateurs, vérificateurs...)

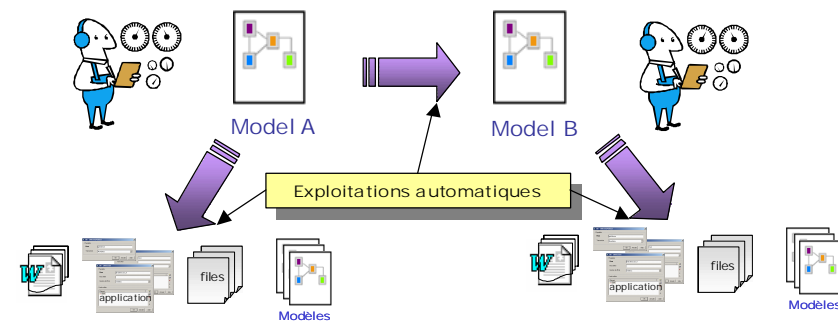
Approche traditionnelle basée sur des documentations non structurées



I Des Composants MD

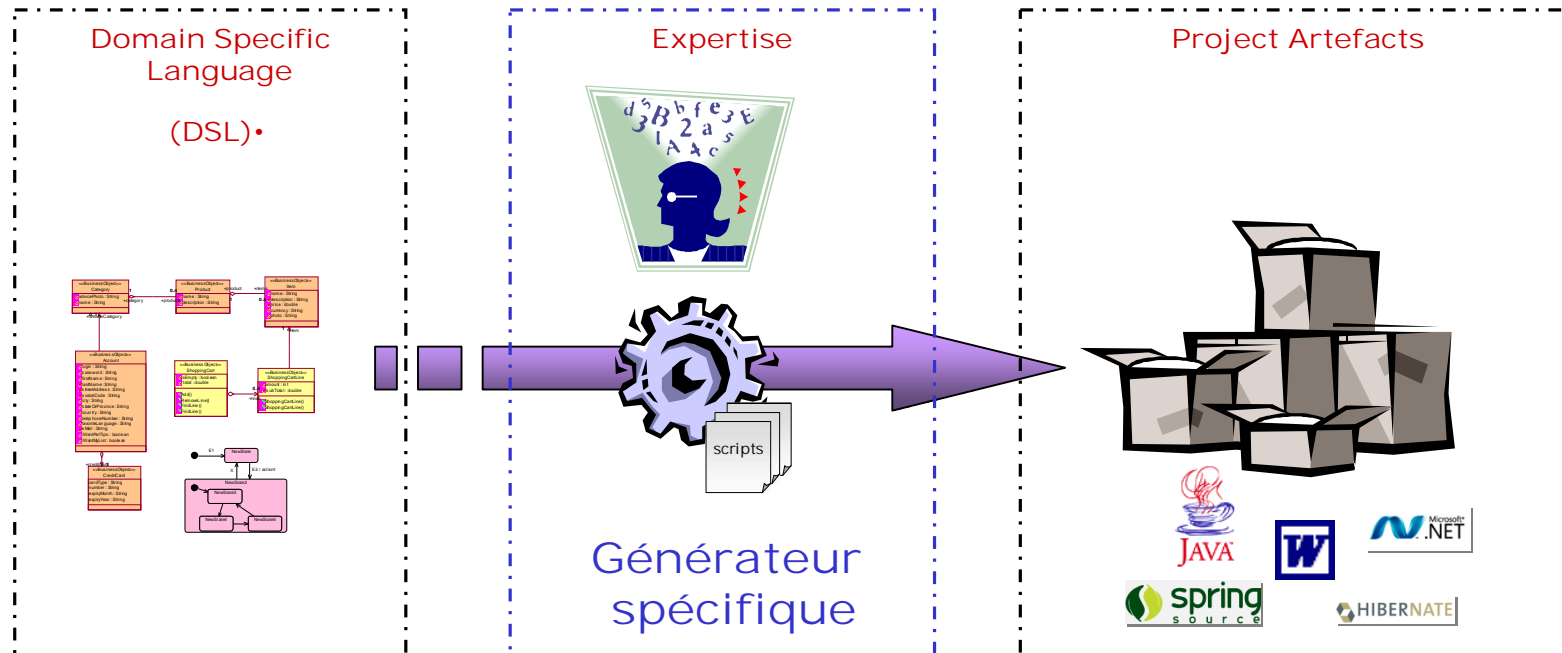
Capitalisation des expertises et des savoir-faire

Généralisation et diffusion de cette expertise



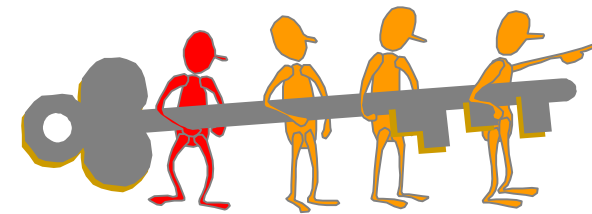
Approche « Model Driven »

Chaine de production MD



MD + 3 Principes clés

- 3 principes d'utilisation :
 - Adaptation
DSLs et générateurs sur-mesure
 - Simplicité
Modèle à Code
 - "Agilité"
DSLs et générateurs en évolution constante



Adaptation (1/2)•

It is better to have tools that fit the domain, rather than tools that require the domain and organization to change...

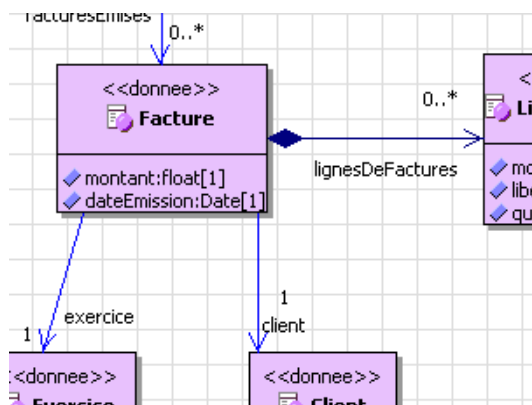
- Utilisation de DSLs (Domain Specific Language)•

Langage spécifique = simple, efficace et adapté à un problème

Les bonnes métaphores

- Exemples de domaines de modélisation :
 - Objets métiers
 - IHM et cinématique
 - MonFramework : Les termes qui-vont-bien pour décrire mon application à base de composants distribués, par exemple

Mise en oeuvre avec des profils UML ou des langages textuels



```

Application Library {
  basePackage = org.library

  Module media {

    Service LibraryService {
      findLibraryByName => LibraryRepository.findLibraryByName;
      findMediaByName => MediaRepository.findMediaByName;
      findMediaByCharacter => MediaRepository.findMediaByCharacter;
      findPersonByName => PersonService.findPersonByName;
    }

    Entity Library {
      scaffold
      String name key
      - Set<@PhysicalMedia> media <-> library
    }

    Repository LibraryRepository {
      findByQuery;
      @Library findLibraryByName(String name) throws LibraryNotFoundException;
    }
  }

  Entity PhysicalMedia {
    scaffold
    String status length="3"
    String location
  }
}
  
```

Adaptation (2/2)•

- Composants MD « sur mesure » qui exploitent les modèles DSL
 - Générateurs (de code, de doc, de métriques, ...)
 - Vérificateurs, simulateurs, transformateurs
- Le code généré peut être complété manuellement

```

«DEFINE Start FOR Model»
  «IF isFeatureSelected("JPAAnnotations")»
    «EXPAND PersistenceConfig»
  «ELSE»
    «EXPAND HibernateProperties»
    «EXPAND HibernateCfgXml»
  «ENDIF»
«ENDEFINE»

«DEFINE HibernateCfgXml FOR Model»
«FILE "hibernate.cfg.xml"»<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>

    <!-- Database connection settings -->
    <property name="connection.driver_class">«getConnectionDriverClass()»</property>
    <property name="connection.url">«getConnectionUrl()»</property>
    <property name="connection.username">«getConnectionUsername()»</property>
    <property name="connection.password">«getConnectionPassword()»</property>

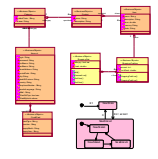
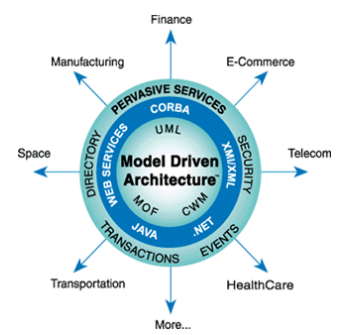
    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">«getConnectionPool_size()»</property>

    <!-- SQL dialect -->
    <property name="dialect">«getDialect()»</property>

```

Simplicité

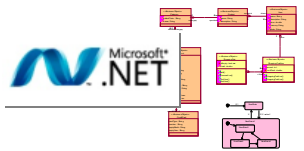
MDA



Modèle simple
Indépendant
de la plateforme technique



Transfo de modèle
à Merge de modèle...



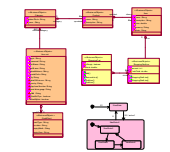
Modèle très technique,
Délicat à gérer...



Ce qu'on peut obtenir
d'un modèle technique
et d'un générateur
généraliste



MD + 3 principes



Modèle simple
(DSL) •
Architecture Logique



Générateur
spécifique



Tout ce dont
on a besoin...



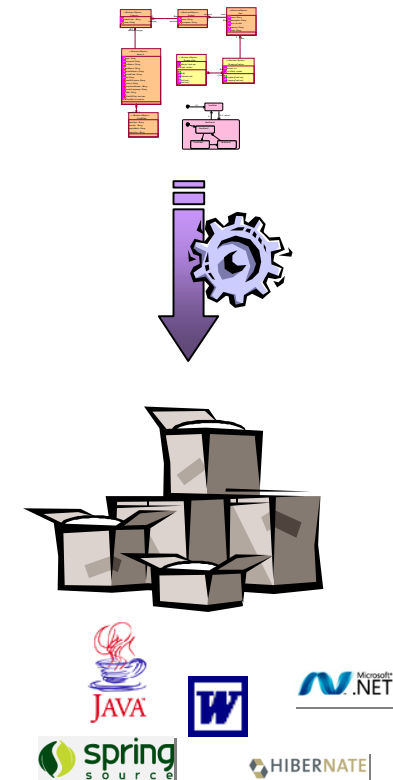
“Agilité”

- Développer et faire évoluer conjointement le générateur et l'application
- Démarche pour l'ajout d'une capacité :

Prototyper une solution

Enrichir les profils UML si nécessaire

Coder la solution une fois dans le générateur



Cartographie des concepts agiles

- Cartographie des pratiques => eXtreme Programming

Valeurs

- Communication
- Simplicité
- Feedback
- Courage
- Respect

Principes

- Feedback rapide
- Assumer la simplicité
- Changements incrémentaux
- Accueillir le changement
- Un travail de qualité
- Expérimentation concrète
- Faible investissement de départ
- Jouer pour gagner
- Communication ouverte et honnête
- Responsabilités acceptées
- Adaptation aux conditions locales
- Voyager léger
- Mesures honnêtes

Pratiques

- Planification itérative
- Livraison de petites releases
- Utilisation de métaphores
- Conception simple
- Tests
- Refactoring du code
- Programmation en binôme
- Propriété collective du code
- Intégration continue
- Rythme soutenable
- Client sur site
- Standards de codage

Communication

- Tous les principes agiles restent utiles et applicables
post-its, time-boxing, tempêtes de cerveaux, tableaux blancs, etc.
- Les choix sont formalisés dans le modèle
- Les métaphores sont incarnées par les DSLs
- Le modèle n'est pas seulement une documentation. Il sera automatiquement transformé en code
Documentation valorisable, exploitable
- Fondements de la modélisation : faciliter la communication
boîtes et flèches, psycho cognitive et de résolution de pbs•

Conception / refactoring / changement

- è La génération de code en masse permet des changements importants (voire radicaux) avec peu d'effort
- è Changements incrémentaux, sur deux axes parallélisable :
 - Incréments fonctionnels (modèle)
 - et incréments techniques (générateur)•
- è Le générateur permet de démultiplier les efforts de développement
- è Modifications répercutées automatiquement à tous les artefacts d'un projet

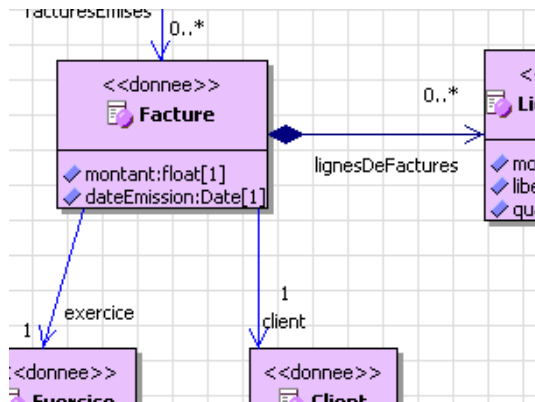
Feedback

Le logiciel et les tests sont produits plus rapidement

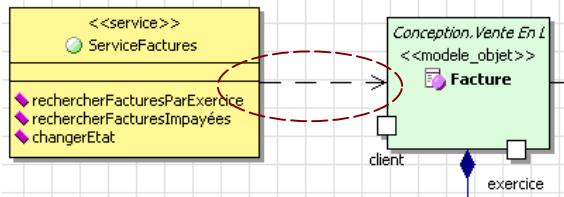
MD permet de capitaliser de projet en projet (profils et générateurs réutilisables)•

- Permet de montrer très rapidement au client un logiciel exécutable
 - ex: générer un prototype instantanément avec différents framework standard (Struts, JSF, Spring, Hibernate etc)•

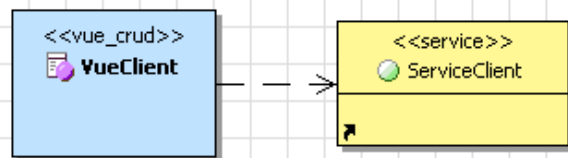
Feedback - Exemple



Modélisation des données

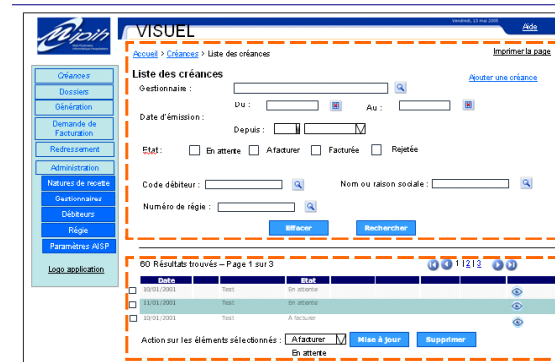


Modélisation des services

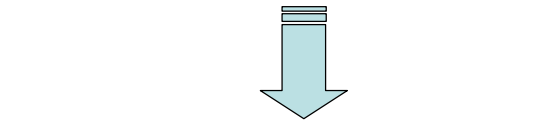


Modélisation des IHMs

3 boîtes et 2 flèches suffisent à générer une application directement exécutable... (au moins autour des services CRUD).



Panneau de recherche



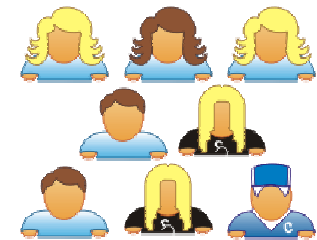
Panneau de détails

Travail de qualité, standards de codage

- è Code de qualité généré en masse
- è La modélisation avec DSL favorise l'expression de solutions plus simples, mieux comprises et donc potentiellement mieux conçues
- è Validation automatique des modèles
- è Généralisation automatique des meilleures pratiques
- è Tous les artefacts générés à partir du modèle restent synchronisés (code, documentation, etc.) sans effort.

Propriété collective du code

- Les générateurs et les DSLs font partie du code
- Ne pas s'en faire une montagne :
 - Générateur = un programme comme un autre
 - DSL = un profil UML ou une grammaire
 - Les développeurs sont leurs propres experts métier pour établir les profils !
- De nouveaux savoir-faire à acquérir...
 - Langage de template
 - Profil UML
 - ...



Faible investissement de départ

- Le MD peut être mis en œuvre progressivement.
- Le MD crée de la valeur rapidement.
- Investissement vite rentabilisé

Temps de développement d'un template :

- Fichier simple : x 3 le fichier à la main (ex, fichier de propriétés, config...)
- Fichier compliqué : x10 ? (jsp, statemachine...)

⇒ROI atteint :

- cas 1 : si plus de 3 fichiers générés
- cas 2 : si plus de 10 fichiers générés

Conclusions

- Investissement et risques limités
- Bénéfices nombreux et importants
- Compatibilité avec les principes des démarches agiles

what else... ;-?