From Model Driven Development to Model Driven Architecture

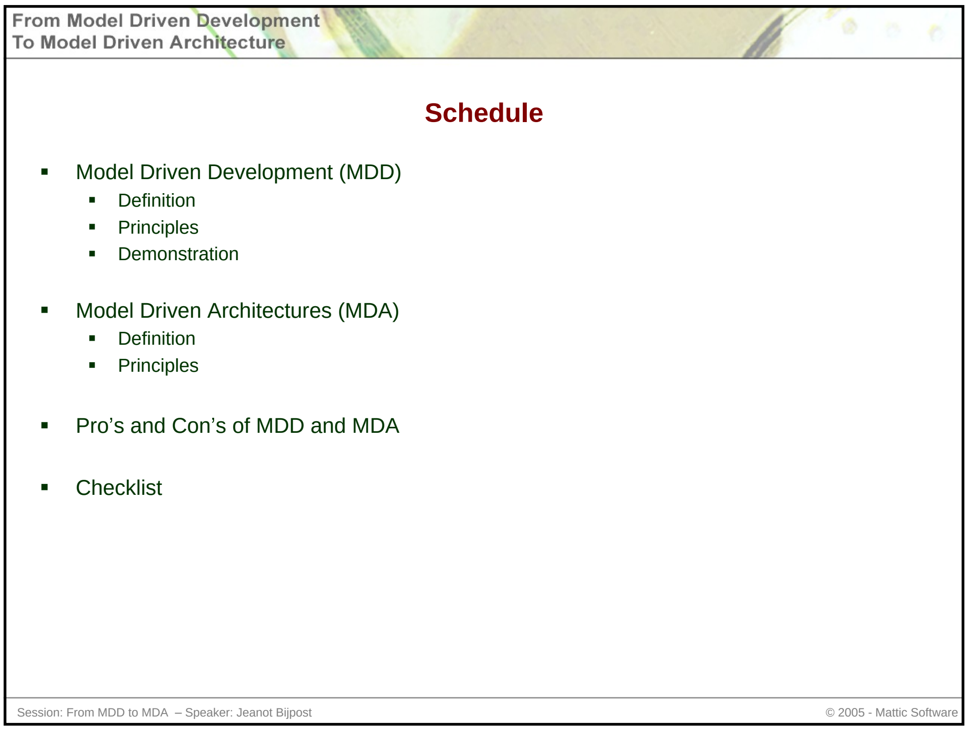
Firebird Conference 2005

Speaker: Jeanot Bijpost / Mattic Software

Mattic® software

Duration of the original presentation: 50 minutes.

The slides on the upper side of the pages were used during the presentation.
The slides on the lower side contain additional comment.



Schedule

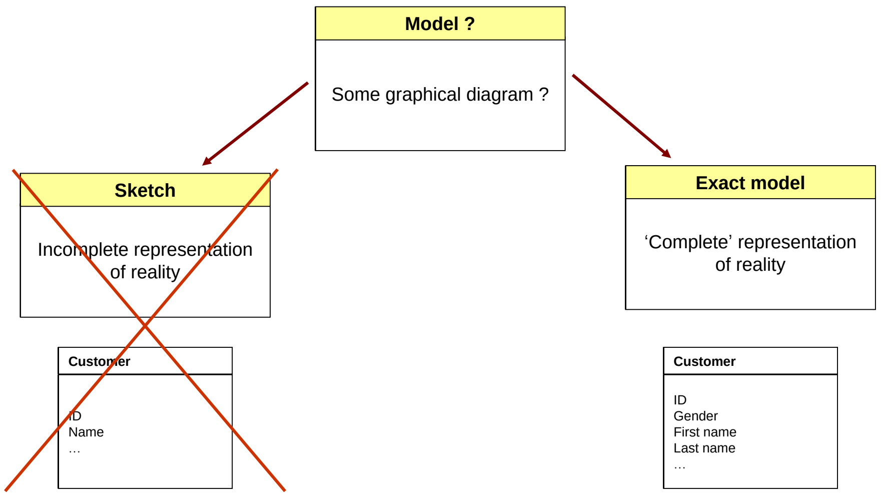
- Model Driven Development (MDD)
    - Definition
    - Principles
    - Demonstration
- Model Driven Architectures (MDA)
    - Definition
    - Principles
- Pro’s and Con’s of MDD and MDA
- Checklist

History of programming languages

1st Generation	Machine Language
2nd Generation	Assembly Language
3rd Generation	Procedural languages Object oriented languages
3rd Generation / **4th Generation**	Application Frameworks Database Access Frameworks
4th Generation	Application/Code Generators Domain specific languages

- MDD tools are often seen as the 4th generation of programming languages.
- The definition of the 1st, 2nd and 3rd generation are quite clear.
- The 4th generation is however less clear.
  - If we have a 3rd generation language including sophisticated frameworks to create interfaces and access databases and this environment offers us code generators and visual designers… is it still just a 3rd generation language or is it a 4th?
  - What is the difference between such an environment and a MDD enviroment?

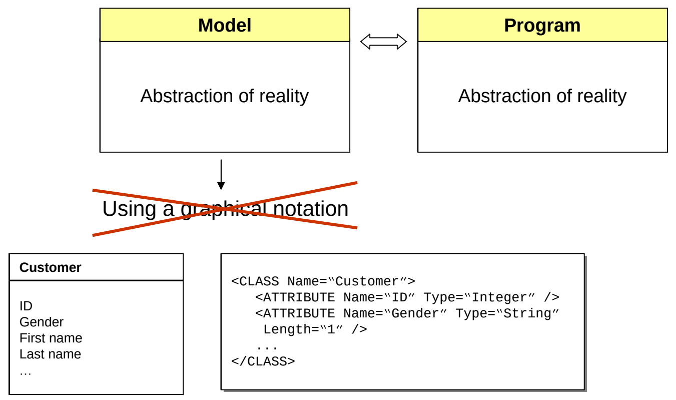
Model ?



Let's start with the word 'Model'.

- Most people associate a model with a diagram, a graphical notation.
- Models can be used in different ways:
- As a sketch
  - Used by programmers to communicate about programs.
  - Such a sketch is often incomplete. When we draw a UML class on a whiteboard we will only draw what is relevant for the discussion at hand.
- As an exact model
  - Used by MDD generators to generate a database or application (or code to create one).
  - Like a program an exact model should be complete. It should contain every relevant aspect.

Model v.s. Program



```
<CLASS Name="Customer">
   <ATTRIBUTE Name="ID" Type="Integer" />
   <ATTRIBUTE Name="Gender" Type="String"
    Length="1" />
   ...
</CLASS>
```

Abstract v.s. complete

- When we study the movement of the planets using a model, this model should be **complete** regarding to information such as mass and velocity. However we do not need to know the temperature of the planets, we **abstract** from this kind of information.
- A model is always an abstraction of the reality.
- So how does this compare to a program? There is no difference! Both a model and a program are abstractions of reality.

So what is the difference? The graphical notation?

- Consider the UML class on the left and the XML code on the right.
- Both describe the same information.
- So is a graphical notation required to describe a class?
- No! It is merely the syntax (notation). It is not a fundamental difference.

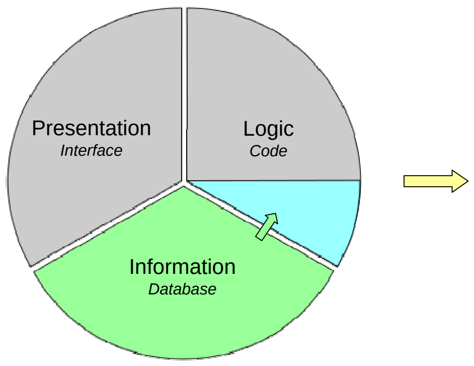
Model Driven Programming (MDD) v.s. 'Normal' Programming

'Normal' Programming		Model Driven Development
• Graphical and non-graphical languages • Compiler / Virtual machine	⟺	• Graphical and non-graphical languages • Code Generator / Compiler • Model Compiler • Interpreter / Virtual Machine • **Higher level of abstraction**

Again: what is the difference between MDD and 'normal' programming?

- One could say 'normal' programming languages do not use diagrams and MDD tools mix both diagrams and textual notations. Is this true?
  - This might be true for older languages. Modern IDE's use visual UI designers and some environments even integrate UML diagrams.
- Normal programming languages use a compiler. This compiler delivers either executable code or code for a virtual machine. How is this done in MDD?
  - Different MDD tools use different techniques. Code generation and compilation, direct compilation of a model, interpretation, it is all there.
  - So we finally found thé difference!? Not very convincing is it…
- Apart from differences in IDE's and notations the only fundamental difference seems the level of abstraction. MDD simply uses a higher level of abstraction to define a program.
  - (Just like Java uses a higher level of abstraction compared to assembler).

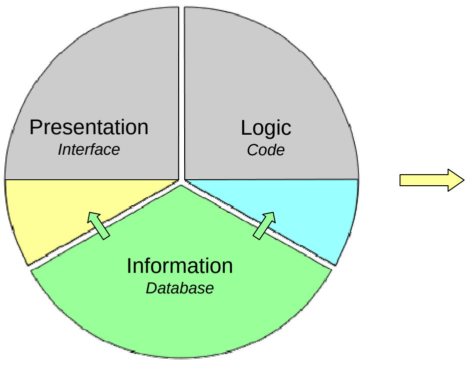
Model Driven Development – ‘Object Oriented’



- Code Generation
- Classes
- Getters/Setters
- Empty methods

- For years and years MDD was closely related to database development. In the past few years MDD can also be found in object oriented environments. Although this presentation focuses on databases a discussion about MDD would be incomplete without taking a peek at object orientation.
- In OO a (UML) class model can be used to generate a skeleton for your code. This skeleton consists of class declarations with:
  - Generated getters and setters to access the attributes.
  - Empty method declarations (ready to be completed with non-generated code).
- More and more languages offer support for this kind of generation. In some environments your code can be ‘reverse engineered’ into a class diagram.
- Some environments are capable of generating more, such as:
  - The SQL-DDL code to create a database to implement the required persistency.
  - A complete layer with classes to access this database.
  - A complete layer with base classes to implement your logic.
- Although there are a lot of questions regarding to topics such as transactions and locking in these environments, they are getting closer and closer to the functionality offered by the traditional MDD tools.
- In the end the main difference might be the starting point. Traditional MDD uses the information model (data model) whereas OO uses the class model.

Model Driven Development – 'Data Oriented'



- **Insert / Update / Delete**
- **Search**
- **Master-Detail**
- **Basic Help**

- Traditional MDD starts with a model of your database. Using this model MDD generators can create default screens/applications with:
  - Insert / Update / Delete.
  - Search (including operators and nested search).
  - Each default screen will display the details of the main table.
  - Basic help system (generated from field descriptions and from descriptions of basic interface components).

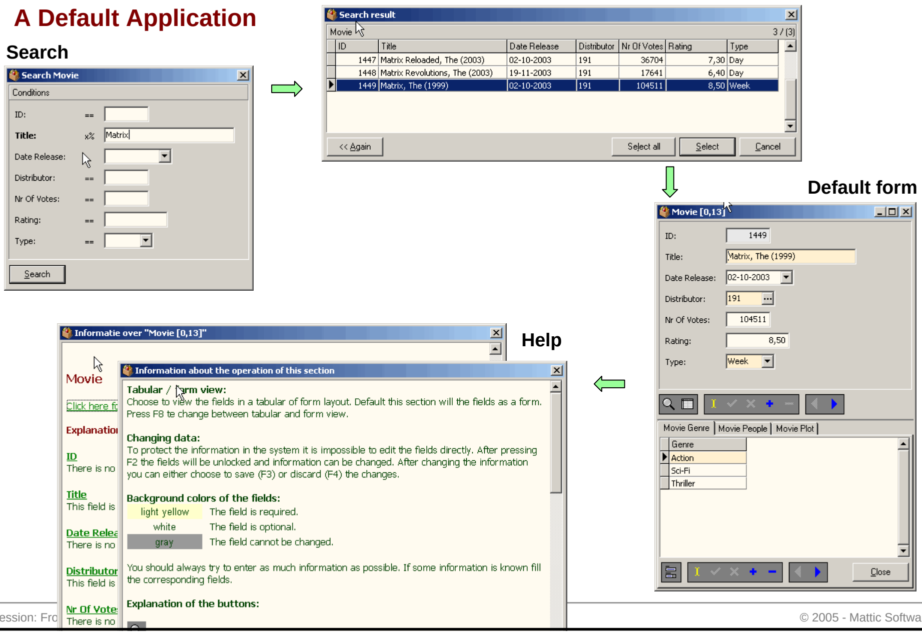
A Default Application

Search

Search result

Default form

Help



Search Movie form:

ID:	==	
Title:	x%	Matrix
Date Release:		
Distributor:	==	
Nr Of Votes:	==	
Rating:	==	
Type:	==	

Search result — Movie 3 / (3)

ID	Title	Date Release	Distributor	Nr Of Votes	Rating	Type
1447	Matrix Reloaded, The (2003)	02-10-2003	191	36704	7,30	Day
1448	Matrix Revolutions, The (2003)	19-11-2003	191	17641	6,40	Day
1449	Matrix, The (1999)	02-10-2003	191	104511	8,50	Week

Movie [0,13]

ID:	1449
Title:	Matrix, The (1999)
Date Release:	02-10-2003
Distributor:	191
Nr Of Votes:	104511
Rating:	8,50
Type:	Week

Movie Genre | Movie People | Movie Plot

Genre
Action
Sci-Fi
Thriller

Informatie over "Movie [0,13]"

Movie

Click here fo

Explanatio

ID
There is no

Title
This field is

Date Relea
There is no

Distributor
This field is

Nr Of Vote
There is no

Information about the operation of this section

Tabular / form view:
Choose to view the fields in a tabular of form layout. Default this section will the fields as a form. Press F8 te change between tabular and form view.

Changing data:
To protect the information in the system it is impossible to edit the fields directly. After pressing F2 the fields will be unlocked and information can be changed. After changing the information you can either choose to save (F3) or discard (F4) the changes.

Background colors of the fields:

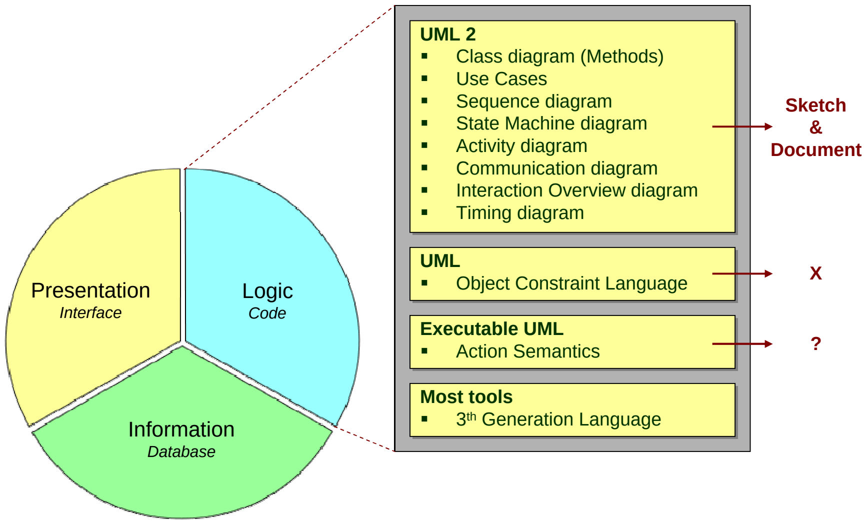
light yellow	The field is required.
white	The field is optional.
gray	The field cannot be changed.

You should always try to enter as much information as possible. If some information is known fill the corresponding fields.

Explanation of the buttons:

- The screens were generated directly from the structure of the database. Not a single line of handwritten code has been written, no properties were set of changed.
- There seem to be two types of generation:
  - One time only generation, and
  - Repeatable generation
- With repeatable generation you can change your model, regenerate and still keep your customizations.
- “One time only generation” can be considered as “a Model Driven Start”.
- “Repeatable generation” can be considerd as “Model Driven Development”.
  - Since development is more that just getting started.
- So we now have a two fold definition:
  - More abstract when compared the third generation languages
  - Repeatable generation
- For more information about Cathedron:
  - Take a look at the session “An introduction of Cathedron”.
  - Visit www.cathedron.com.

Modeling logic ?



UML 2
- Class diagram (Methods)
- Use Cases
- Sequence diagram
- State Machine diagram
- Activity diagram
- Communication diagram
- Interaction Overview diagram
- Timing diagram

UML
- Object Constraint Language

Executable UML
- Action Semantics

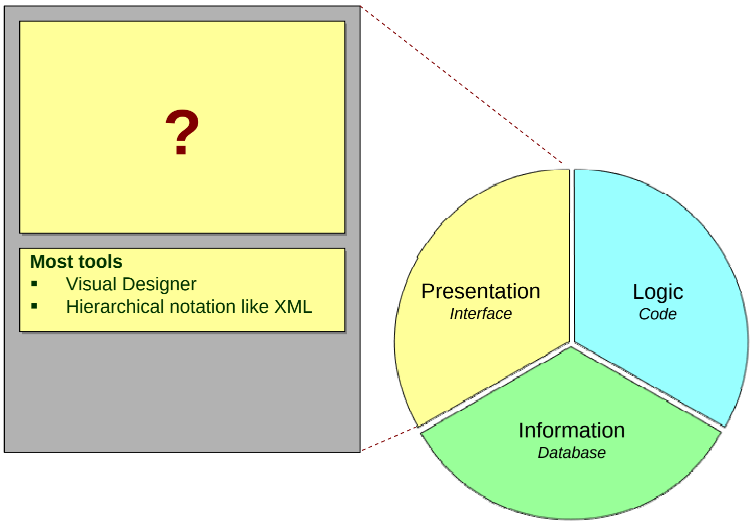
Most tools
- 3th Generation Language

- So far we have only been using a database diagram (ERD/UML) to generate a default application.
- How about logic? Is there any way to model logic? What is used by MDD tools?

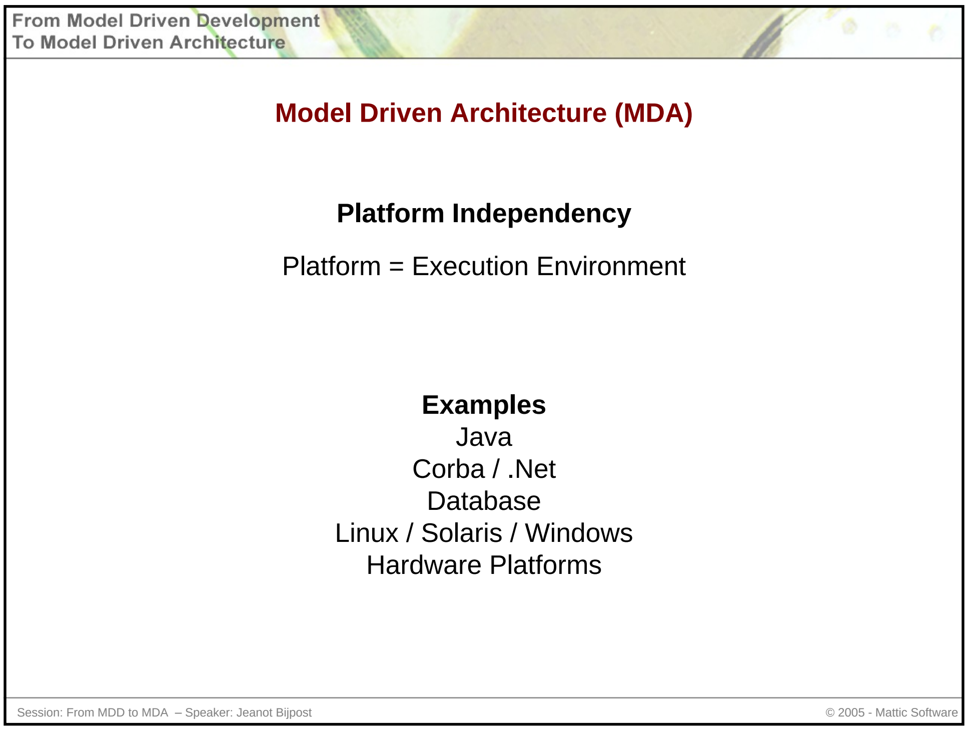
- The options:
  - UML offers a broad range of diagrams to model both the workflow and the more standard logic of your application. We think UML will not become the next generation visual language to model your code. The diagrams overlap and consistency checking is a nightmare (and missing from (most?) tools).
  - UML offers the object constraint language (OCL). However this language is not powerful enough to create the logic you normally require.
  - In Executable UML the action semantics language is used. This language abstracts from the way sets/lists are implemented. Although the syntax seems rather focused on object orientation the language might be capable of specifying a query for a database as well. Will this be the next generation language???

- The present:
  - Most MDD tools simple use a 3rd generation language.

Modeling interfaces ?



- How about the interface? Is there any way to model an interface? What is used by MDD tools?
- There seems to be no standard ‘interface definition language’.
- Every vendor seems to implement its own language. These languages do look very similar. They all use a hierarchical notation (like xml) to define interface components and their properties.

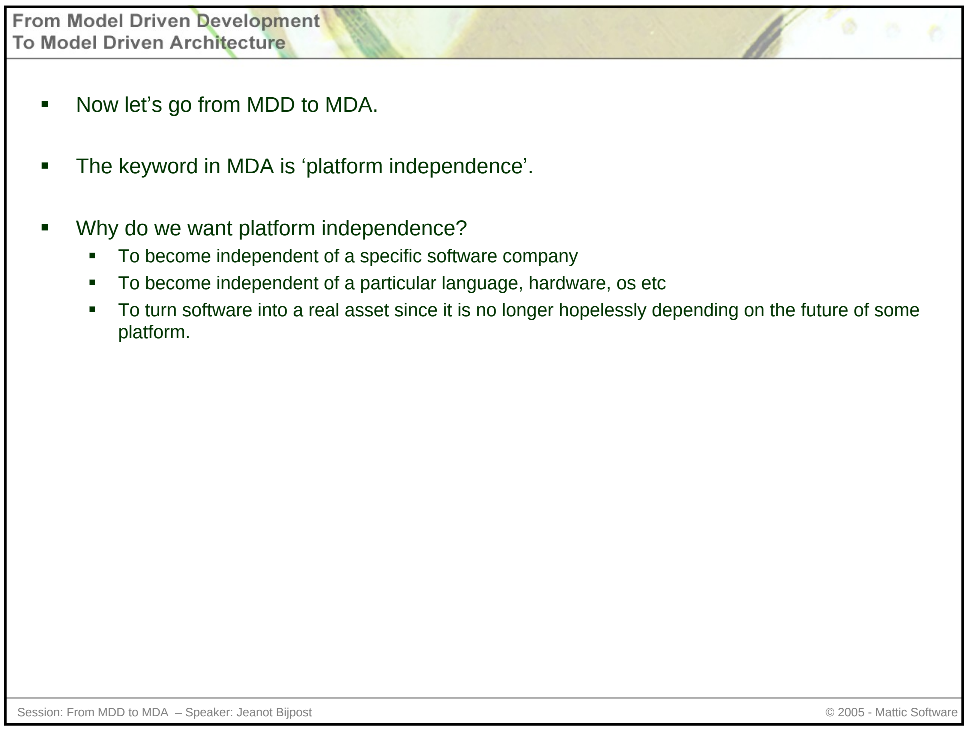


Model Driven Architecture (MDA)

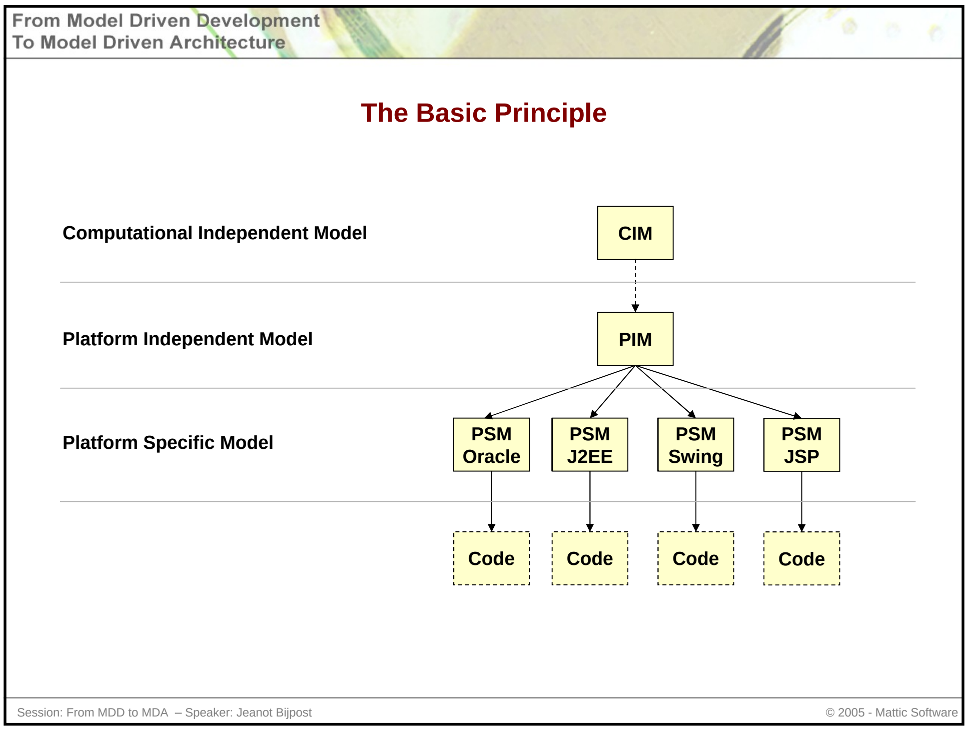
Platform Independency

Platform = Execution Environment

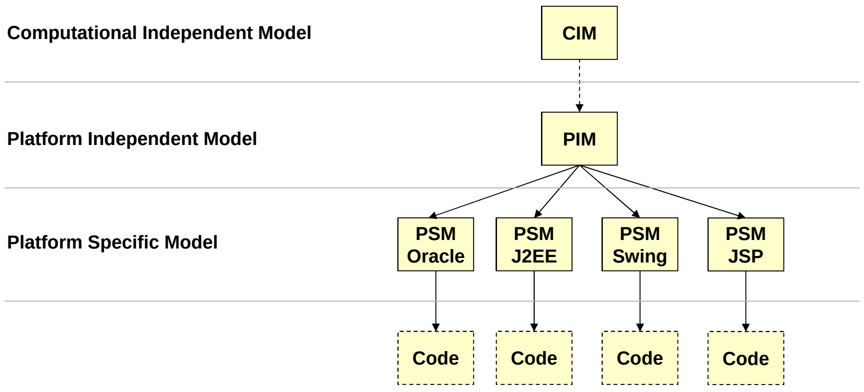
Examples
Java
Corba / .Net
Database
Linux / Solaris / Windows
Hardware Platforms



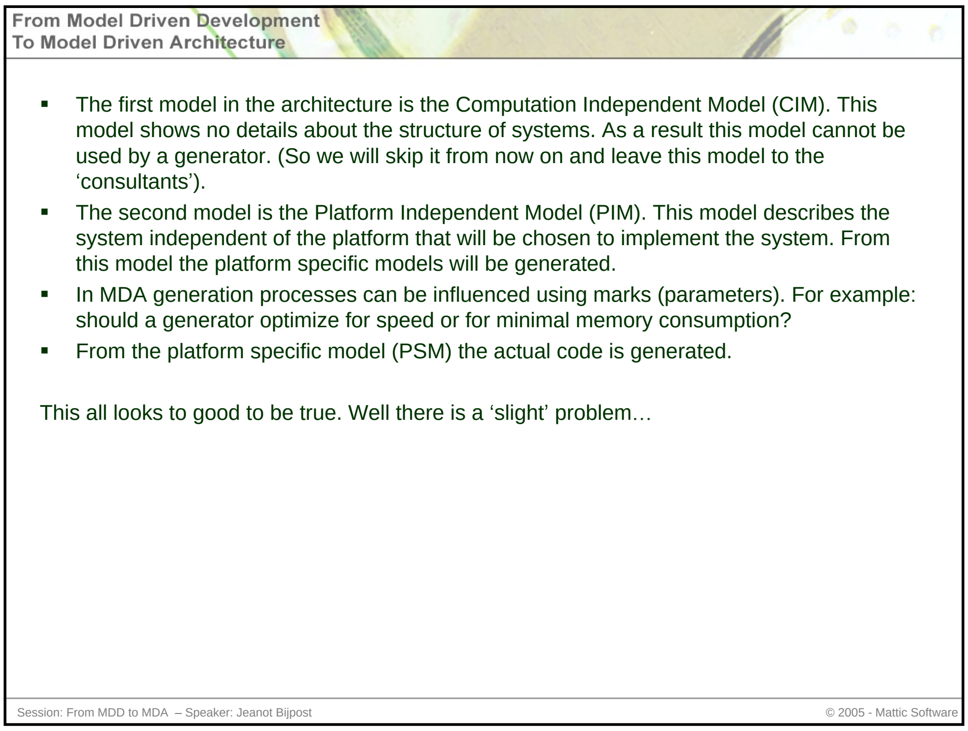
- Now let’s go from MDD to MDA.
- The keyword in MDA is ‘platform independence’.
- Why do we want platform independence?
  - To become independent of a specific software company
  - To become independent of a particular language, hardware, os etc
  - To turn software into a real asset since it is no longer hopelessly depending on the future of some platform.



The Basic Principle



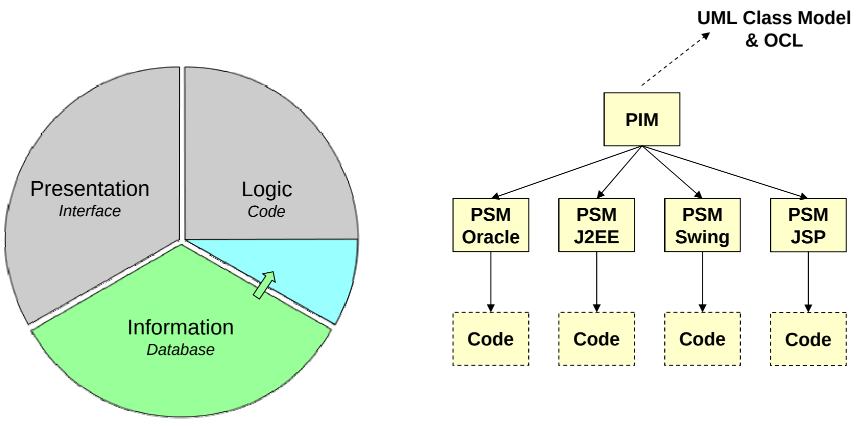
$$\begin{tikzcd}
\text{Computational Independent Model} & & & \boxed{\text{CIM}} \arrow[d, dashed] \\
\text{Platform Independent Model} & & & \boxed{\text{PIM}} \arrow[dl] \arrow[dll] \arrow[d] \arrow[dr] \\
\text{Platform Specific Model} & \boxed{\text{PSM Oracle}} \arrow[d] & \boxed{\text{PSM J2EE}} \arrow[d] & \boxed{\text{PSM Swing}} \arrow[d] & \boxed{\text{PSM JSP}} \arrow[d] \\
 & \text{Code} & \text{Code} & \text{Code} & \text{Code}
\end{tikzcd}$$



- The first model in the architecture is the Computation Independent Model (CIM). This model shows no details about the structure of systems. As a result this model cannot be used by a generator. (So we will skip it from now on and leave this model to the 'consultants').
- The second model is the Platform Independent Model (PIM). This model describes the system independent of the platform that will be chosen to implement the system. From this model the platform specific models will be generated.
- In MDA generation processes can be influenced using marks (parameters). For example: should a generator optimize for speed or for minimal memory consumption?
- From the platform specific model (PSM) the actual code is generated.

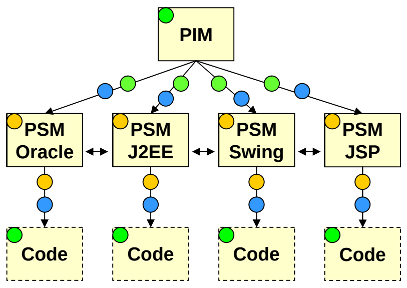
This all looks to good to be true. Well there is a 'slight' problem…

What is modeled ?



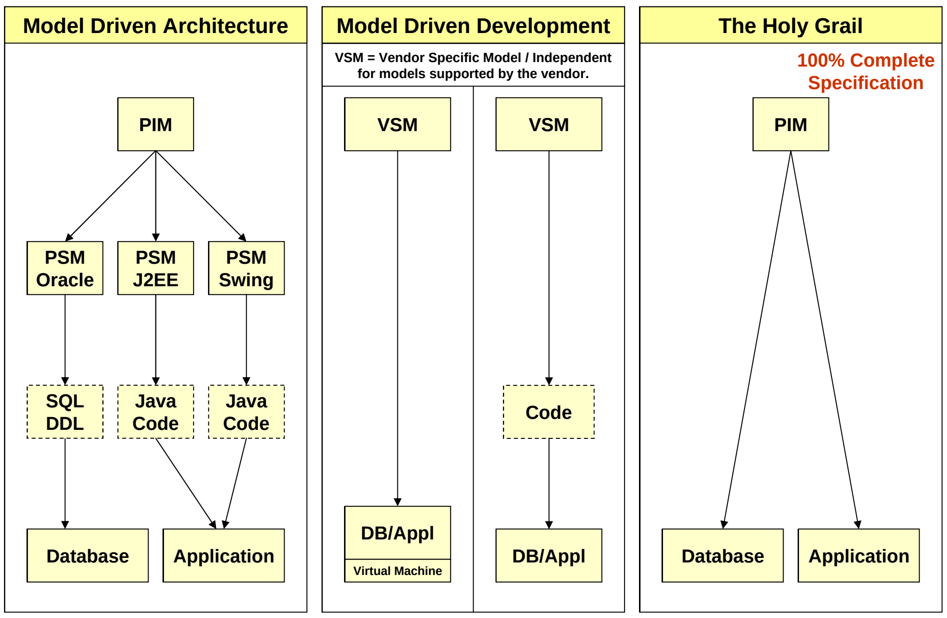
- The PIM in the current generation of MDA tools seems to be specified using only the UML Class Model in combination with OCL!
- As stated before there is no proper language to define your code or to interface.
- As a result you will have to specify the interface and logic at the ‘code’ level.

Where to change ?



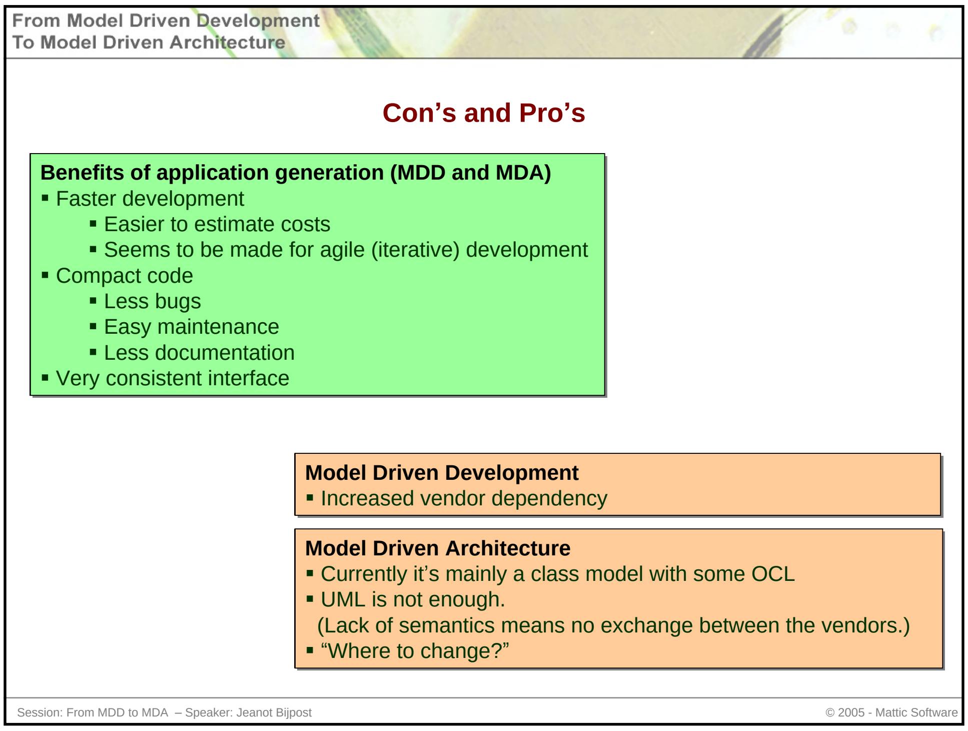
- Another problem with MDA seems to be the numerous points where we can change the definition of our system. We can change:
  - The PIM.
  - The PSM's.
    - Although this is dangerous. If we delete a column from the database PSM how will our Swing PSM know this? MDA defines bridges. So far those bridges do not seem powerful enough to handle such changes.
  - The code.
    - Since the PIM is very limited a lot of changes will end up here. (Being completely platform depended).
- There is more! We could change:
  - The parameters for the generators (Marks).
  - The generators.

Every dot in the diagram above represents a point to change a system.
Powerful or confusing ???



From right to left:

- ‘The Holy Grail’ is a way of developing without PSM’s or code. This would require a PIM-language powerful enough to express 100% of the functionality of a system. (And that’s why it’s a holy grail).
- In MDD we have a vendor specific model (VSM) without an easy way to exchange this model with another tool. As a result we are depending on this vendor.
  - Note: The two columns represent the many different techniques used by MDD tools. (As we have seen before there are more than two techniques).
  - Note: The VSM model can be platform independent! Most MDD vendors support multiple platforms. Again we are depending on the vendor, if a platform is unsupported you can rebuild your system using another tool.
- When MDA grows beyond the ‘UML Class diagram‘ and offers proper ways to specify large portions of your logic and interfaces on the PIM level, things would become interesting. When 80%-90% of our systems can be specified using the PIM we might be independent enough to make a relative painless switch between vendors and platforms. Software might become a real asset.



Con’s and Pro’s

Benefits of application generation (MDD and MDA)
- Faster development
    - Easier to estimate costs
    - Seems to be made for agile (iterative) development
- Compact code
    - Less bugs
    - Easy maintenance
    - Less documentation
- Very consistent interface

Model Driven Development
- Increased vendor dependency

Model Driven Architecture
- Currently it’s mainly a class model with some OCL
- UML is not enough.
  (Lack of semantics means no exchange between the vendors.)
- “Where to change?”

Checklist

- **Do you need multiple platforms ?**
- **What is the lifetime of your application ?**
- **Do your colleagues accept an application generator ?**

- **What is generated ?**
  (A code skeleton / Code for a working application / A working application using a virtual machine)
- **Is the generation fast enough for my way of working ?**
  (How agile is your development process ?)
- **How do I specify: information / presentation / logic / workflow ?**
  (And how platform independent is this specification ?)
- **Is the entire specification used by the generator ?**
  (Or is it partially for documentation purposes only?)
- **Is there a way to check the consistency between the models ?**
- **How can I integrate special code/components ?**
  (reports, custom screens, interfaces to special hardware such as scanners etc)
- **If you can add custom code: what happens to this code when you regenerate?**
- **How about version control?**