Model-Driven Software Development: What it can and cannot do

Jon Whittle

Associate Professor

Information & Software Engineering

George Mason University

http://ise.gmu.edu/~jwhittle

Outline

- Introduction to Modeling
- Introduction to OMG's Model-driven Architecture (MDA):
  - What is MDA?
  - Example
  - MDA supporting technologies: metamodeling, transformations, executable UML
  - Tool support
- Introduction to Microsoft's Software Factories
  - Domain Modeling
  - Domain-Specific Languages
  - The Microsoft-OMG Debate
- Model-Driven Development (MDD) in the future

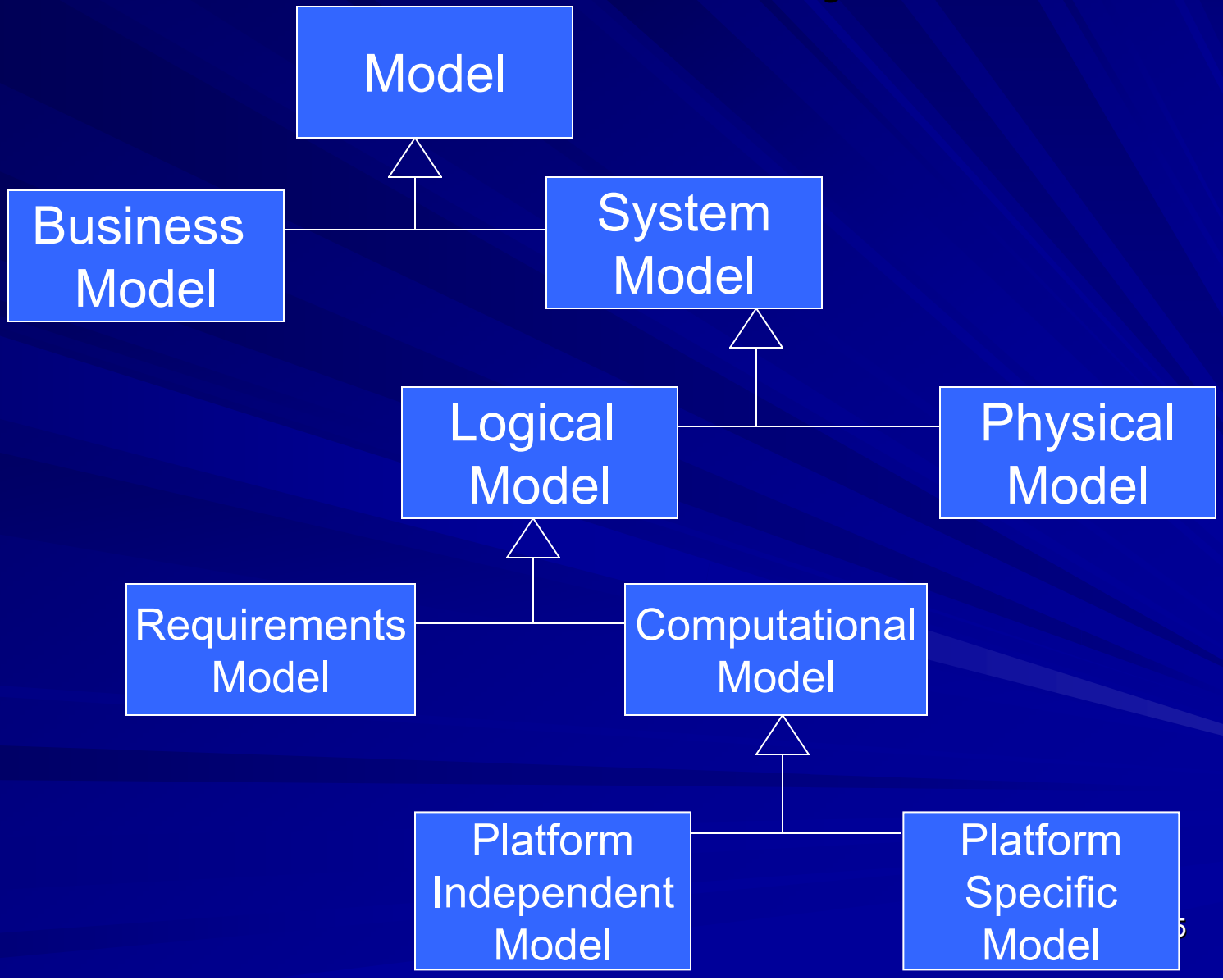
System Modeling

- What is a (system) model?
  - *"A simplified description of a complex entity or process"* [web dictionary]
  - "*A representation of a part of the function, structure and/or behavior of a system*" [ORM01]
  - "*A description of (part of) a system written in a well-defined language*" [KWB03]
- Key point:
  - ***Models are abstractions***
  - Entire history of software engineering has been one of raising levels of abstraction (01s → assembly language → 3GLs → OO → CBD → patterns → middleware → declarative description)

Why Model?

- Models can be used in:
  - System development
  - System analysis
  - System testing/validation/simulation
- *Each requires abstraction of complexity*
- Why not model?
  - Large(r) effort required
  - Synchronization
  - Delayed return
  - Requires specialized skills

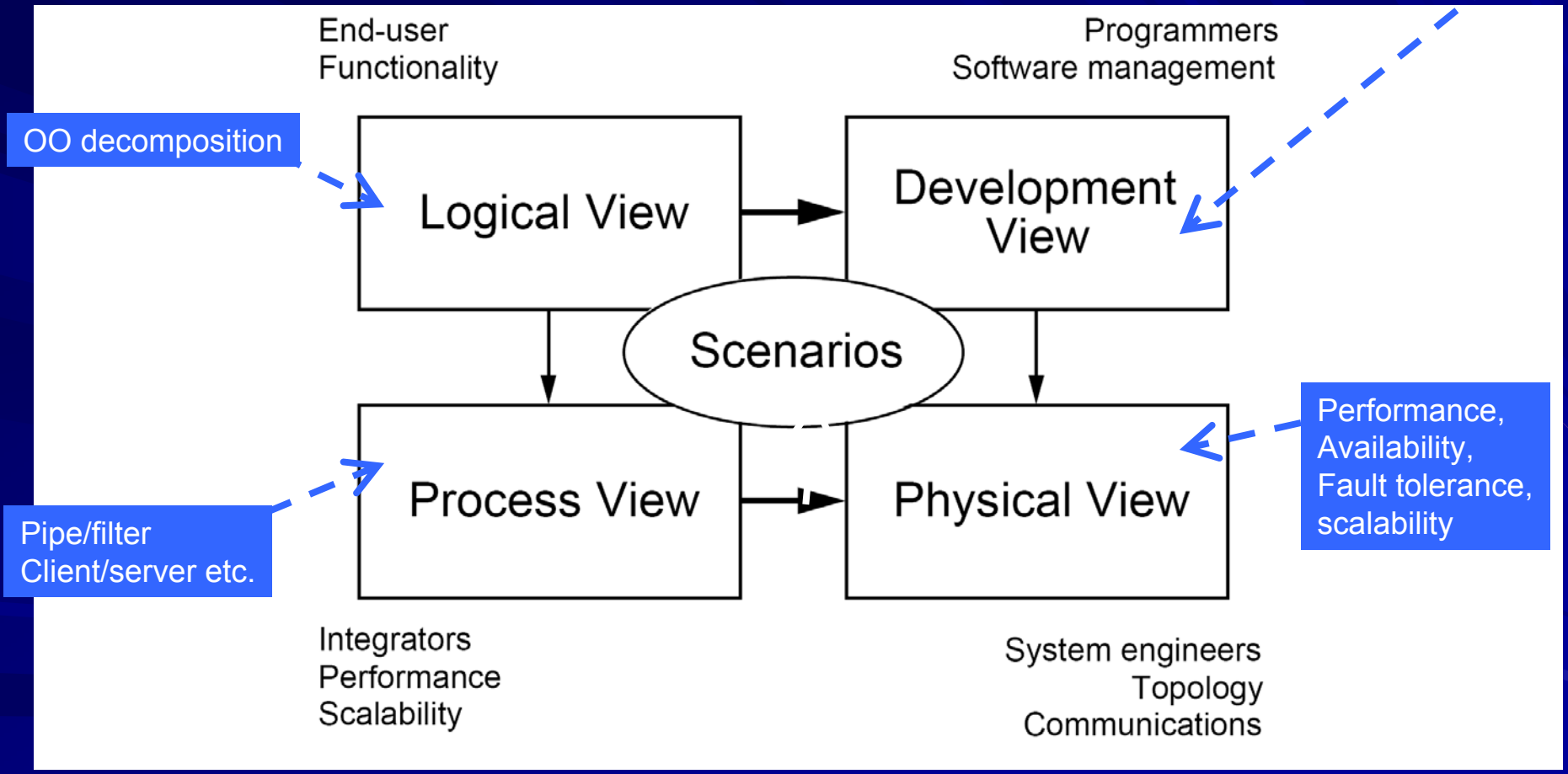
Model Taxonomy



From FRA03

Cf. 4+1 views

Internal reqmts:
Cost/planning, schedule,
Monitoring, reuse, portability,
Etc. Layered view.



From [4+1]

Abstraction in UML

- Use case model abstracts away:
  - Computation
  - Irrelevant or low-priority functionality
  - Non-functional requirements
- Analysis model abstracts away:
  - Decisions on “ilities”: performance and distribution requirements; optimizations etc.
  - Decisions on data structures
  - Booch/Rumbaugh/Jacobson: 1:5 ratio going from analysis to design
  - (for class diagrams): operations interface, “design attributes”, visibilities, navigability
- Design model abstracts away:
  - Physical deployment (nodes, processors etc.)
  - Use of Source files, libraries etc.

Abstraction in UML (cont'd)

- Sequence diagrams abstract to:
  - Inter-object (component, process) communications
- State machines abstract to:
  - Intra-object behaviors
  - Without method implementations
  - Without middleware concerns
  - Abstract communication
- Class diagrams abstract to:
  - Structure, no behavior

UML2.0

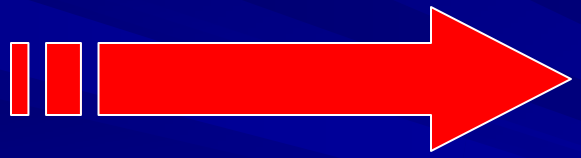
- What's new:
  - Support for components
  - Much improved interaction diagrams
    - MSC-like constructs
    - Interaction overview diagrams
  - Metamodel for OCL2.0
  - More consistent UML metamodel
  - Tighter relationship between MOF and UML

- Modeling software systems:
  - What is the future?

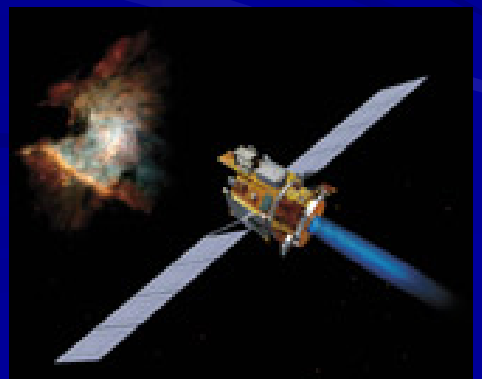
Sketch

Blueprint

Programming language



Rhapsody, RoseRealTime, Simulink,
Rational XDE, OptimalJ



DS1:
230,000LOC

Outline

- Introduction to Modeling
- Introduction to OMG's Model-driven Architecture (MDA):
  - What is MDA?
  - Example
  - MDA supporting technologies: metamodeling, transformations, executable UML
  - Tool support
- Introduction to Microsoft's Software Factories
  - Domain Modeling
  - Domain-Specific Languages
  - The Microsoft-OMG Debate
- Model-Driven Development (MDD) in the future

MDA Definition

- Model Driven Architecture:
  - Recent initiative from the Object Management Group (OMG) making models and transformations between models first-class elements
  - Builds on UML and OCL (Object Constraint Language)
  - More info:
    - *MDA Explained: The Model Driven Architecture, Practice & Promise,* Anneke Kleppe, Jos Warmer & Wim Bast
    - *Model Driven Architecture. Applying MDA to Enterprise Computing.* David S. Frankel. OMG Press 2003.
    - http://www.omg.org

Why MDA?

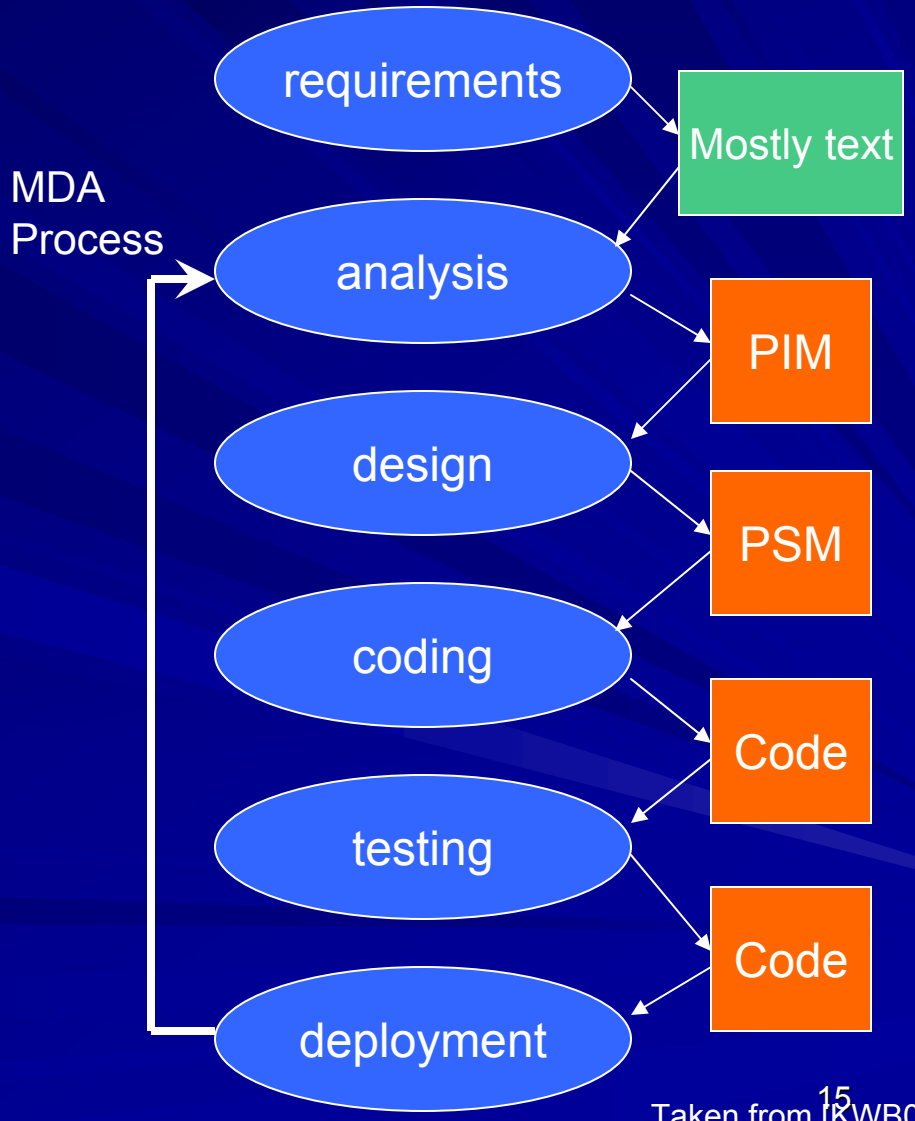
- Productivity problem:
  - Time/budget-crunch means early lifecycle artifacts are not maintained
    - Requirements documents, design documents etc. (UML or other) become obsolete
  - Why spend so much time on high-level specs anyway?
    - Cf. eXtreme programming, agile modeling
- Portability problem:
  - Technology is growing fast (Java, XML, UML, J2EE, .NET, JSP, SOAP, Flash etc.)
  - How does software keep up?

Why MDA?

- Interoperability problem:
  - Systems need to interoperate with other systems
    - Cf. web-based systems, component-based systems
- Maintenance/Documentation problem:
  - Despite best-practice advice, most software projects do not adequately model/document software, let alone maintain it

MDA Development Lifecycle

- MDA lifecycle similar to traditional lifecycle, but:
  - Emphasis on creation of formal models (i.e., models that can be understood by computer)
- 2 types of models:
  - *Platform-independent model*
  - *Platform-specific model*
- 3 types of transformations:
  - *PIM-to-PIM*
  - *PIM-to-PSM*
  - *PSM-to-code*
- Importance of transformation automation



Taken from [KWB03]

MDA Benefits

- **Productivity**:
  - Work is done mainly at the PIM level
  - PIM-to-code transformation is automated
  - Caveat: need to define transformations, but they can be reused (hopefully)
- **Portability**:
  - PIMs are implementation-independent
  - Incorporate new technologies by defining new (reusable) transformations
- **Interoperability**:
  - MDA tools should generate *bridges*
- **Maintenance/Documentation:**
  - Changes to the PIM can be filtered down to PSM/code by re-applying transformations

MDA requirements

- Consistent & precise high-level models
- Standard, well-defined language(s) to write high-level models
  - OMG Standards
    - Modeling – UML; Metamodeling – MOF; Action semantics; Model interchange – XMI; Human-readable textual notation – HUTN; …
- Transformations:
  - Definitions
  - Transformation language (formal) QVT
  - Transformation tools

Models in MDA

- MDA models:
  - Must be *precise (enough)*
  - May be software or business models
  - May be written in UML or some other language
  - May be code (code is a model)
- Caveats:
  - How precise is UML?
  - Is there a clear distinction between a PIM and a PSM?

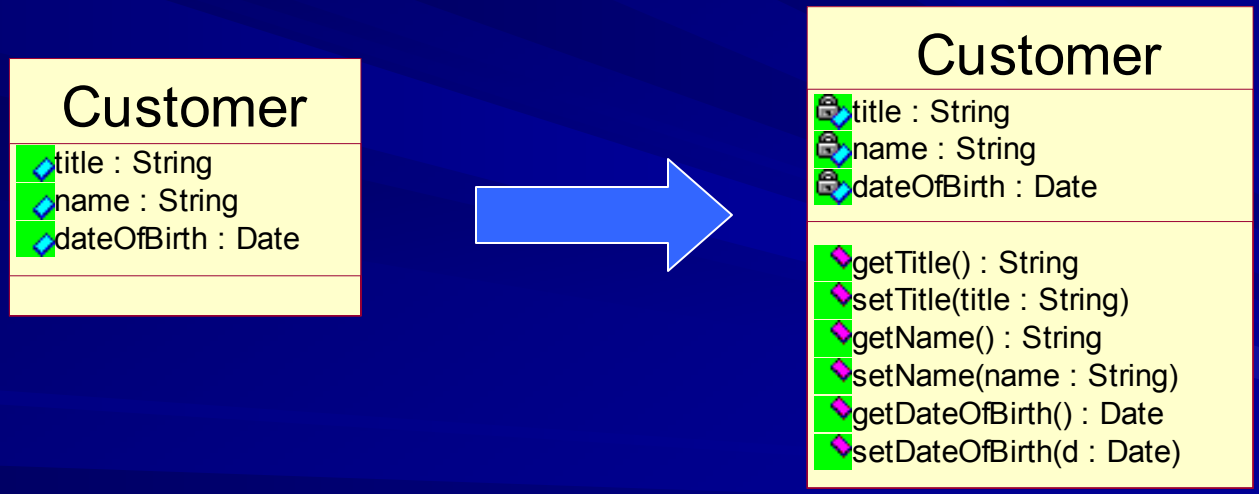
What is a transformation?

- Transformation consists of:
  - A set of transformation *rules*
    - Unambiguous specifications of the way that (part of) one model can be used to create (part of) another model
- Definitions:
  - *A transformation is the automatic generation of a target model from a source model, according to a transformation definition*
  - *A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language*
  - *A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language*
- Preferred characteristics:
  - A transformation should be semantics-preserving
- Note: transformations may be between different languages. In particular, between different *dialects* of UML (UML profiles)

Taken from [KWB03]

Simple Example: public/private attributes

- Transformation: UML PIM to UML/Java PSM
- Purpose: transform public attributes into get/set operations



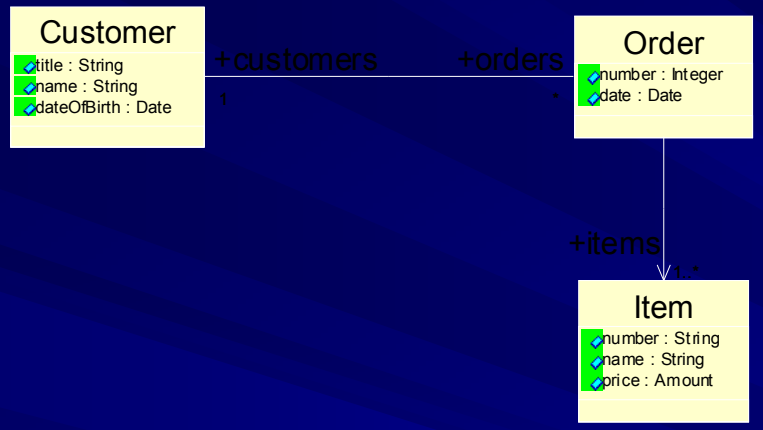
Taken from [KWB03]

Transformation Definition

- For each class named *className*, transform into a class named *className*
- for each public attribute *attributeName : Type* of class *className*, create attributes/operations for *className* in the PSM as follows:
  - Private attribute *attributeName : Type*
  - Public operation *getAttributeName() : Type*
  - Public operation *setAttributeName(att : Type)*
- Exercise: define a Java PSM-to-code transformation that generates Java code for Java PSMs

Taken from [KWB03]

Simple example: associations



Question: is this transformation reversible?

Customer

- title : String
- name : String
- dateOfBirth : Date
- orders : Set
- getTitle() : String
- setTitle(title : String)
- getName() : String
- setName(name : String)
- getDateOfBirth() : Date
- setDateOfBirth(d : Date)
- getOrders() : Set
- setOrders(o : Set)

Order

- number : Integer
- date : Date
- customer : Customer
- items : Set
- getNumber() : Integer
- setNumber(n : Integer)
- getDate() : Date
- setDate(d : Date)
- getCustomer() : Customer
- setCustomer(c : Customer)
- getItems() : Set
- setItems(s : Set)

Item

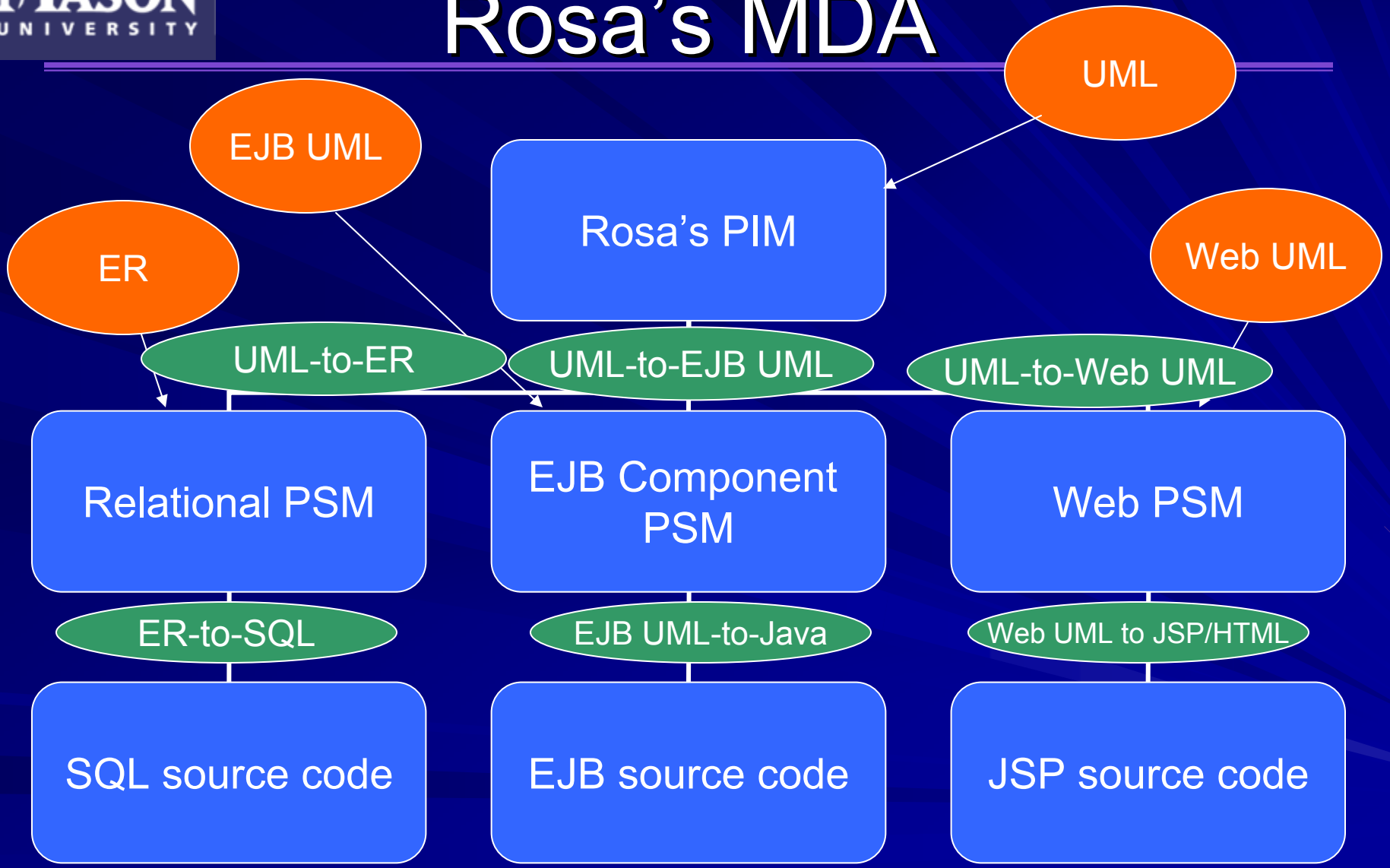
- number : String
- name : String
- price : Amount
- getNumber() : Integer
- setNumber(n : Integer)
- getName() : String
- setName(s : String)
- getPrice() : Amount
- setPrice(p : Amount)

Taken from [KWB03]

More realistic example: Rosa's breakfast service

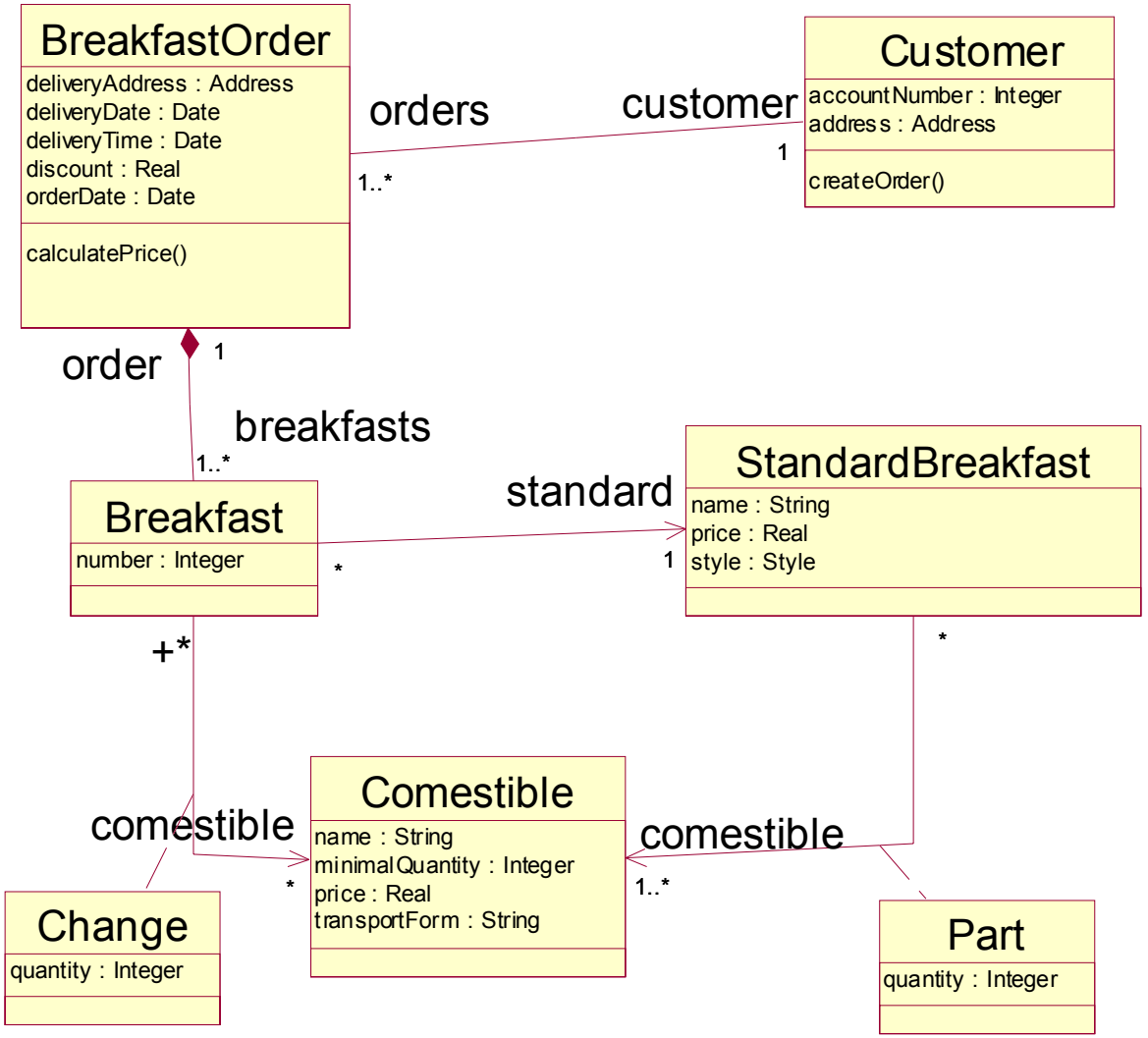
- Example from *MDA explained.* Code downloadable from http://www.klasse.nl/mdaexplained
- Business:
  - Rosa's company home-delivers breakfasts
  - Customers order from web-based menu, choosing time and place of delivery
  - Pay by credit card over the Web
  - Different packages: french breakfast, champagne feast breakfast, indian breakfast
  - Packages are customizable
  - Choice of styles: simple, grand, deluxe
- Software:
  - Web-based, three-tier architecture
    - Database, EJB middle-tier, JSP user interface
  - Two web interfaces: one for customer, one for employees
  - Database of customer details

Rosa's MDA



Adapted from [KWB03]

Rosa's PIM



Taken from [KWB03]

PIM-to-relational transformation

- Consistent object-relational mapping
  - Well-known rules ([2])
- Basic data types:
  - UML String => SQL VARCHAR(40)
  - UML Integer => INTEGER
  - UML Date => DATE
- UML data type (e.g., Address) with no operations => number of columns (one per field of the data type)
- UML class => table
- UML attribute of basic type => column
- UML attribute of class type => use foreign key

PIM-to-relational (contd)

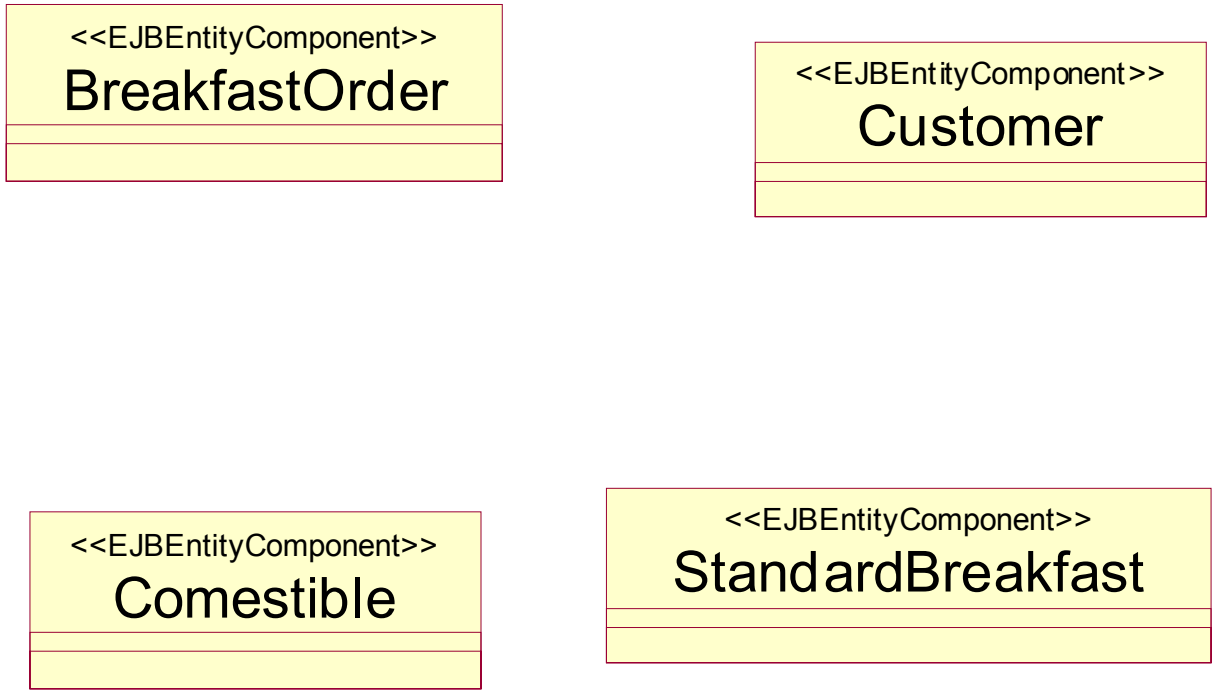
- Association from class A to class B
  - **if** the association A to B is adorned by an association class or the multiplicity at both ends is more than one
  - **then** create a table representing the association class or the association and create foreign keys in both the table representing A and the table representing B referring this new table
  - **else if** the multiplicity at one end is zero or one
    - **then** create a foreign key in the table representing the class at the other end, referencing that end
    - **else** /* the multiplicity of the association is one-to-one */
    - Create a foreign key in one of the tables, referencing the other end

Taken from [KWB03]

PIM-to-EJB Transformation

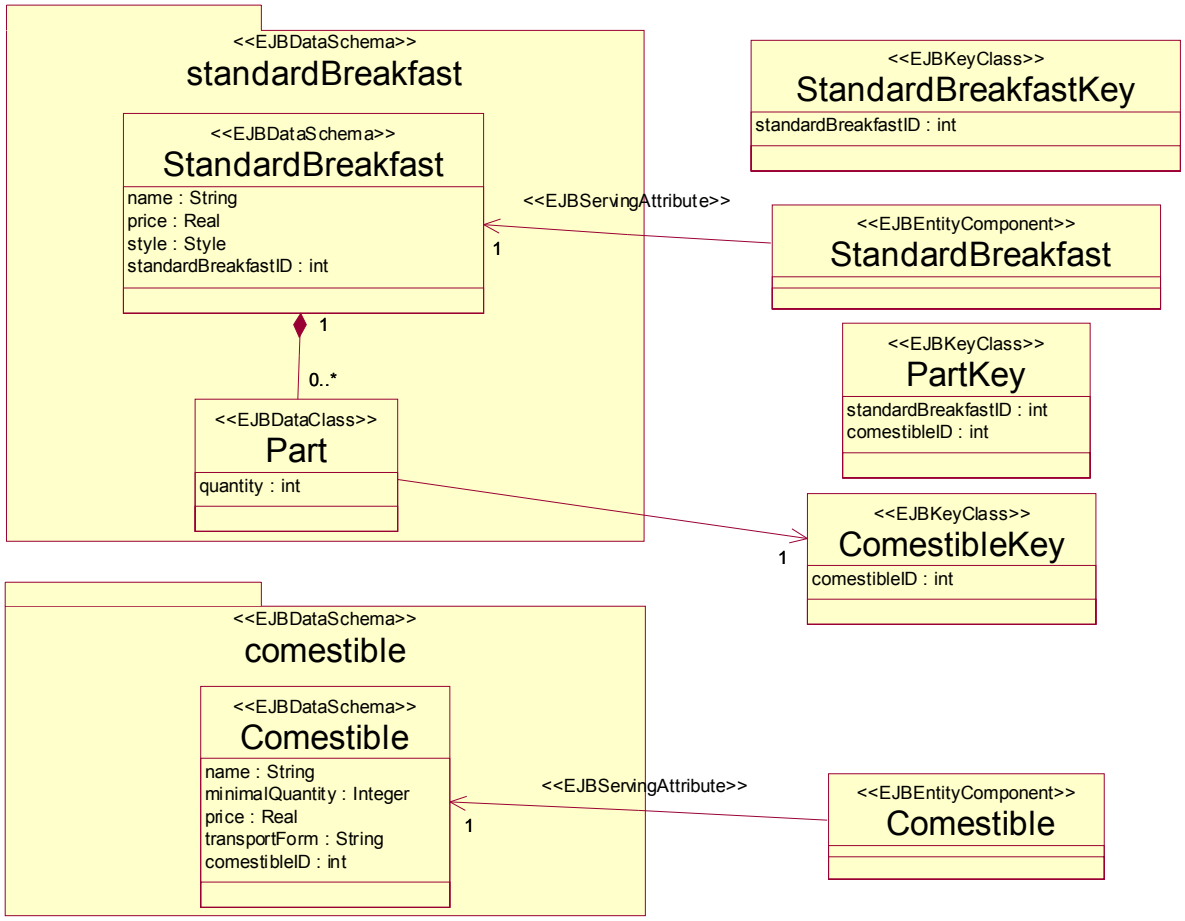
- Key idea: minimize number of components and interactions between components
  - Group related classes into components
  - Exchange data between classes in “chunks”
- This means:
  - *Don’t* create a component for each class
  - *Don’t* create remote method invocations for each get/set-operation
- ***EJB data schema:*** *a set of classes, attributes and associations that is served by an EJB component as a whole*
- ***EJB data class:*** *a class that is part of an EJB data schema*
- Create EJB data schemas using *composite aggregation*:
  - Every class that is part of a whole is clustered into the data schema that is generated from the whole
  - Association classes are clustered into the data schema generated from the associated class that is able to navigate to the other associated class
    - E.g., *Change* becomes part of *BreakfastOrder*

Top-level EJB Components



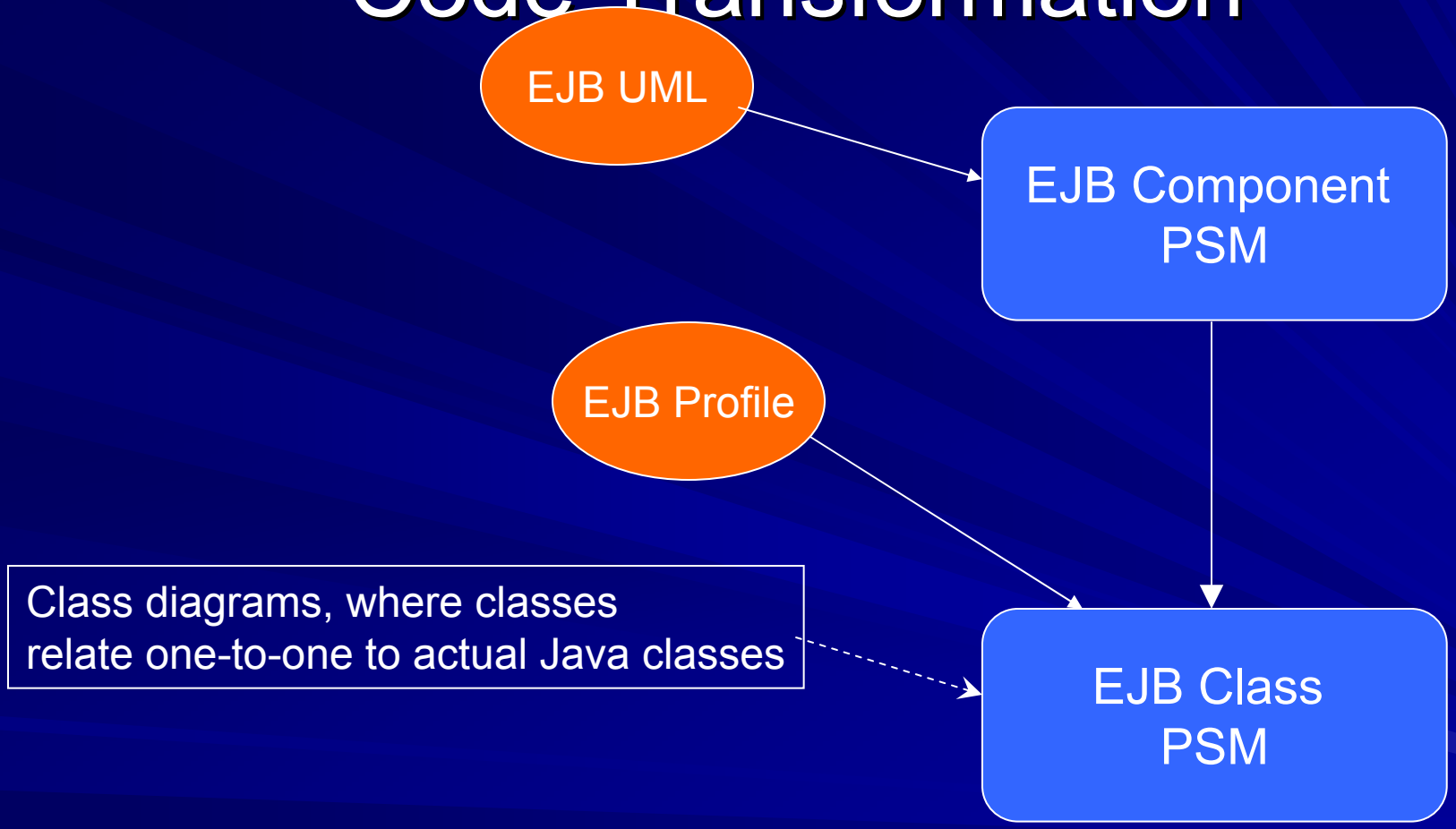
Taken from [KWB03]

EJB Component Model for Comestible and StandardBreakfast



Taken from [KWB03]

EJB Model to Code Transformation



EJB UML → EJB Profile

- To develop entity beans, you must provide:
  - Entity bean class and all classes it depends on
  - Key class
  - Entity bean's remote interface and home interface
  - Deployment descriptor written in XML defining transactional behavior of the business methods and persistence strategy for a component
- Rosa's Breakfast Service will have *container managed persistency*

MDA Supporting Technologies

- Metamodeling
  - How to define your own modeling languages
    - Specializations of UML
    - [Domain-specific modeling languages (DSMLs)]
- Transformations
  - Transformation language
  - Standardization efforts (QVT)
- Making UML more precise
  - Design by Contract
- Executable UML
  - Making UML executable
  - The dawn of the model compiler

UML as PIM

- Plain UML:
  - Strong in modeling structural aspects, weak in modeling behavior
  - Can be used to generate PSMs for structural aspects, but not for behavioral
    - E.g., how would you generate code for use case or interaction diagram?
- Executable UML:
  - Plain UML + dynamic behavior of Action Semantics
  - No standardized concrete syntax
  - State machine becomes anchor point for defining behavior:
    - Each state enhanced with a procedure written in the AS
  - In principle, can be used to generate PSMs for behavior, but:
    - AS too low level – no advantage over writing dynamics of the system in the PSM directly
    - State machines only relevant for certain domains
    - No standardized concrete syntax
- UML/OCL combination:
  - Helps to define precise, unambiguous PIMs
  - Dynamics can be *specified* using pre/post-conditions
  - Still have to write the PSM dynamics, but can check against pre/post-conditions

MDA Tools [based on KY03]

MDA01	Support for PIMs
MDA02	Support for PSMs
MDA03	Can Target Multiple PSMs
MDA04	Model Integration
MDA05	System Evolution
MDA06	Model Interoperability
MDA07	Mappings are modelled
MDA08	Support for Managing Model Complexity
MDA09	Correctness
MDA10	Expressivity
MDA11	Patterns and Genericity
MDA12	Support for Refactoring
MDA13	Intra-Model Mappings
MDA14	Traceability
MDA15	Lifecycle
MDA16	Standardization

Evaluation Scale

Makes things worse	-1
No support	0
Little support	1
Some support	2
Strong support	3
Very Strong support	4

Comparison

Cols 1-4 from [KY03]

		IDE	OptimalJ	Rose RealTime	Objecteering	XMF
MDA01	Support for PIMs	0	4	4	4	4
MDA02	Support for PSMs	1	4	1	4	4
MDA03	Can Target Multiple PSMs	n/a	1	2	3	4
MDA06	Model Interoperability	1	3	3	3	3
MDA07	Mappings are modelled	0	2	0	3	4
MDA09	Correctness	1	3	1	1	1
MDA10	Expressivity	1	4	2	4	4
MDA11	Patterns and Genericity	0	3	2	4	2
MDA12	Support for Refactoring	2	4	2	4	4
MDA14	Traceability	0	1	0	1	2
MDA15	Lifecycle	1	4	4	4	4
MDA16	Standardization	1	4	3	3	4

Productivity Study

- Independent study (Middleware company)
- Development of J2EE petstore application
  - Compuware's OptimalJ
  - Enterprise-level IDE
- Application:
  - E-commerce J2EE petstore
  - 46 page specification
  - Users:
    - sign in and manage their account
    - Browse pet catalog
    - Manage shopping cart
    - Place orders
    - Query orders

Study methodology

- Teams chose their own support tools (e.g., logging, version control)
- 37 scenarios used to test the results to ensure comparability
- Inspection of code to compare quality
- 3 members per team:
  - Senior J2EE architect
  - Experienced J2EE programmers (>3yrs)
- Weekly timesheet submissions on work packages (Estimated vs actual time included)
- 35% increase in productivity for MDA
  - 330 hours vs 508 hours

Why did MDA win?

The value of MDA is analogous to the value of OO in general. It requires more of you in the design phase, and the payoff comes in the implementation phase. And once you've built one or two apps, you really start to get the benefit from it.

It makes brain surgeons better brain surgeons, but it won't make janitors into brain surgeons

- *Less debugging of "silly" mistakes*
- *More efficient (and disciplined) use of design patterns*
- *50-90% of code could be generated*
- *BUT:*
  - *Sometimes, the code generated was very heavy-weight or didn't do what the developers thought it was going to do*
  - *Large learning curve*

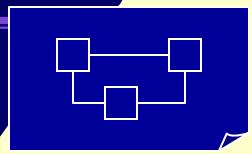
Outline

- Introduction to Modeling
- Introduction to OMG's Model-driven Architecture (MDA):
  - What is MDA?
  - Example
  - MDA supporting technologies: metamodeling, transformations, executable UML
  - Tool support
- Introduction to Microsoft's Software Factories
  - Domain Modeling
  - Domain-Specific Languages
  - The Microsoft-OMG Debate
- Model-Driven Development (MDD) in the future

GEORGE MASON UNIVERSITY

Reusable assets

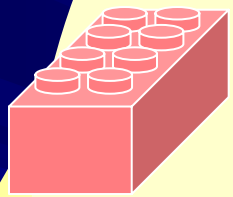
Models



Patterns

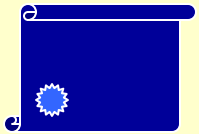
Guidelines

Framework, Components



Architecture

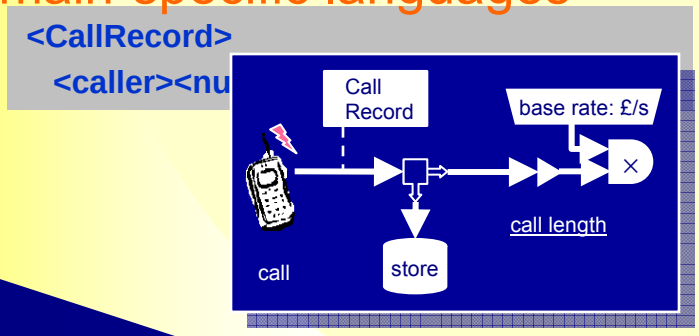
Test suites



Process and roles

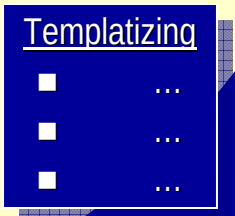
Libraries

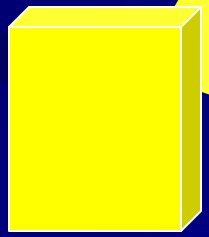
Domain-specific languages



Tools

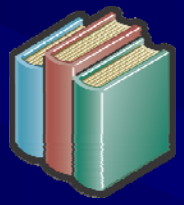
Method & procedures





Automating the Architecture

how to add a service

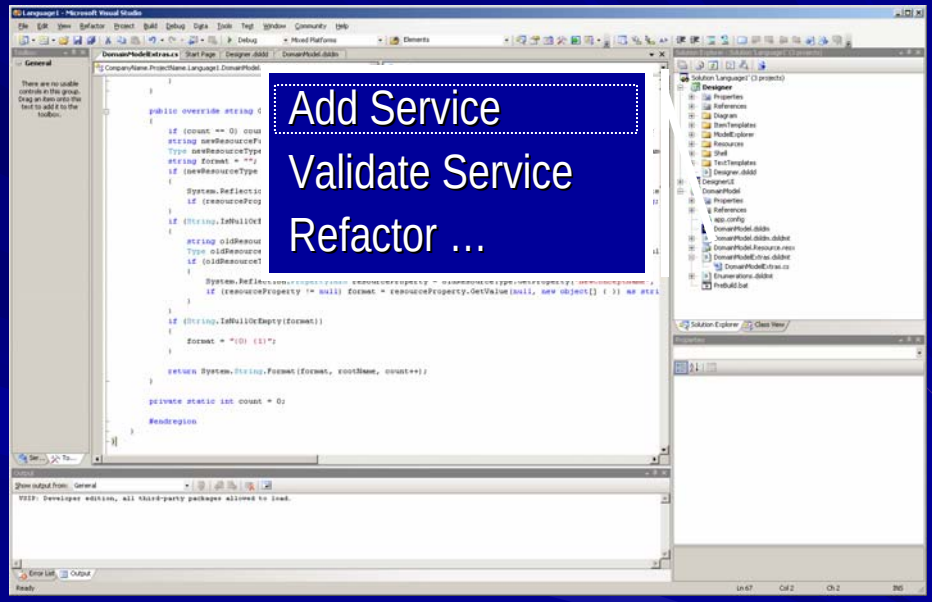


ideas -

conveyed by working together

documented ideas –

not necessarily well-read or up to date!



tooled ideas –

- positive help for developer – saves time
- positive encouragement to conform to architecture

Domain-Specific Modeling

- Language-driven engineering (LDE):
  - Why? *The right abstraction for the right job*
- Metamodel: a model of a language capturing its important properties:
  - Concepts
  - Syntax (Concrete and Abstract)
  - Semantics
- Why?
  - Increased precision
  - Domain-specific mappings
- How?
  - Capture DSML via metamodeling

UML Profiles

- Lightweight approach to tailoring UML
  - E.g. for platforms (.NET, EJB)
  - E.g., for domains (telecomms, real-time)
- Adapt an existing metamodel
  - Cannot change existing metaclasses
- In particular, can only add new constraints
  - Cannot replace existing constraints

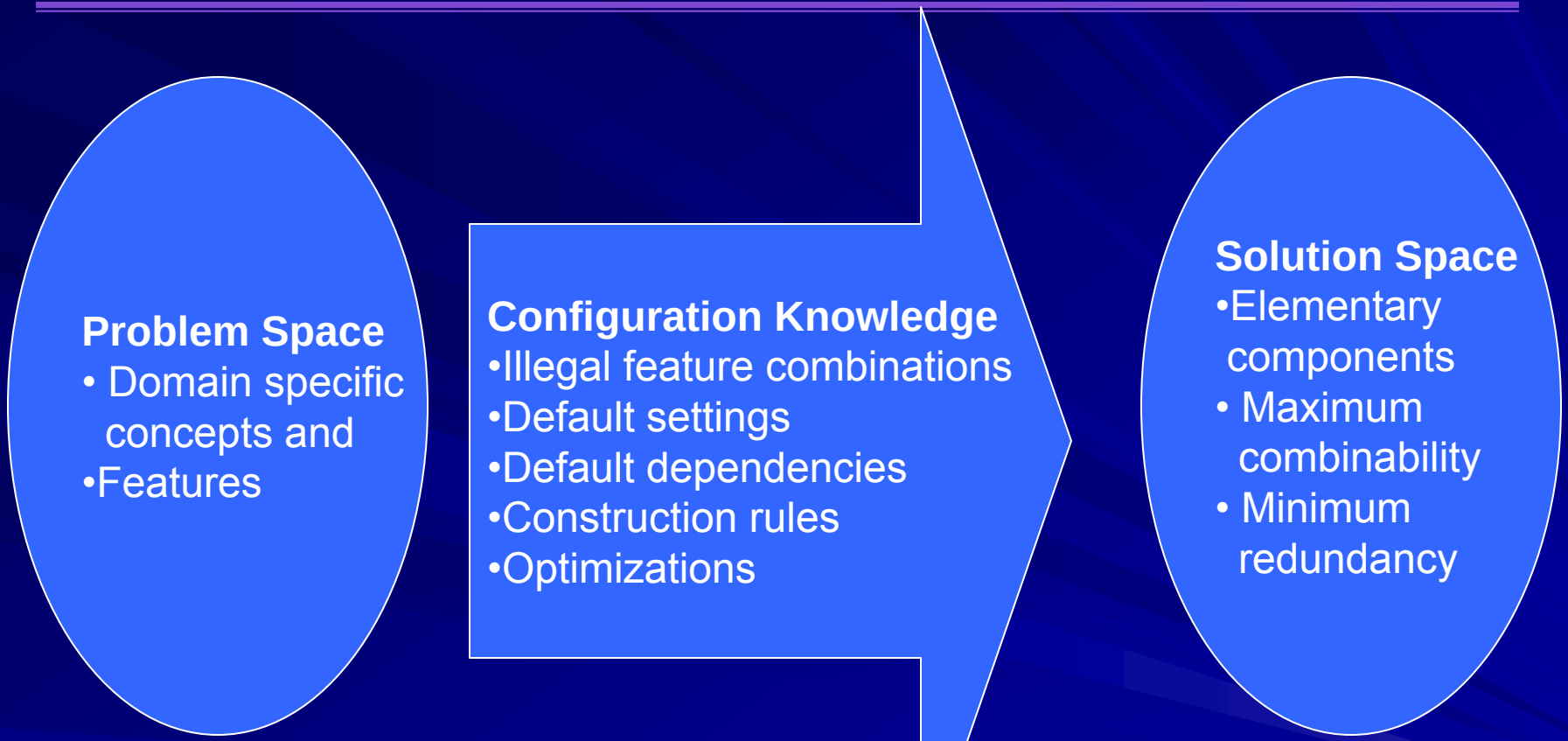
Domain-Specific Modeling

- **Design Patterns**
- **Components**
  - "We deliver the parts, you assemble the car with all the extras"
  - Architectural mismatch
- **Frameworks = Components + Patterns**
  - "We've got the car body, you do the rest..or we'll do it for you for a price"
- **Weak Domain Analysis**
  - Many business application frameworks start off as a development effort for one customer. They grow with more projects.

Generative Programming…

... is a software engineering paradigm based on modeling **software system families** such that, given a particular requirements specification, a highly **customized** and **optimized** intermediate or **end-product** can be **automatically manufactured** on demand from elementary, reusable implementation components by means of **configuration knowledge**.

Generative Domain Model



Domain specific languages
- •Programming Languages
- •Extensible languages
- •Graphical languages
- •Wizards etc.

Generator technologies
- •Simple traversal
- •Templates
- •Transformation systems etc.

Component technologies
- •Component models
- •AOP approaches etc.

Microsoft vs MDA

- Microsoft (Steve Cook):
  - UML is too big and too bad
  - XMI/MOF are just platforms themselves
  - and they change rapidly leading to combinatorial explosion
  - In practice, standards conformance is hard to achieve
- MDA (David Frankel & Michael Guttmann)
  - XMI/MOF problems just due to immaturity
  - DSMLs lead to balkanization
  - Not always easy or cost effective to define DSMLs
  - Microsoft modeling tools are proprietary & map to Microsoft proprietary technology

Outline

- Introduction to Modeling
- Introduction to OMG's Model-driven Architecture (MDA):
  - What is MDA?
  - Example
  - MDA supporting technologies: metamodeling, transformations, executable UML
  - Tool support
- Introduction to Microsoft's Software Factories
  - Domain Modeling
  - Domain-Specific Languages
  - The Microsoft-OMG Debate
- Model-Driven Development (MDD) in the future

MDD: open questions

- How to deal with behavior?
  - State machines?
  - Domain-specific modeling?
- Trust
  - Verifiability of transformations
- Code “tampering”
- Standards evolution (cf. XMI)
- Too many levels of abstraction?
- Social issues
- ROI

MDD Myths

- ***MDD will put programmers out of a job***
  - It doesn't generate 100% code
  - Focus (initially) on "low-hanging fruit"
- ***MDD is just code generation***
  - **it can be code generation, but also model-based testing, simulation, validation, business process modeling etc.**
  - **It is firmly grounded on reusable standards**
- ***MDD model equals UML model***
- ***MDA/MDD is just about OO***
- ***MDD and agile methods don't mix***

Steps to “full-blown” MDA

1. Transition from “sketch” UML to “programming language” UML
2. Identify “killer app” transformations
3. Identify DSML fragments
4. Validate your models
5. Identify “MDA personas”

MDD Research at GMU

- Early lifecycle model simulation
  - Precise requirements language, *use case charts*
  - Grounded in UML but executable
  - Simulate requirements and analysis models early in the loop
- Aspect-oriented modeling
  - Separate development concerns at model time
  - Automatically combine concerns for holistic inspection, analysis, code generation
- Trusted model transformations
  - How can I trust code generated from my model?
- Integrating auto-coding and legacy code/in-house applications
  - Customizing code generators to fit existing architectures

References

- [KWB03] *MDA Explained. The Model Driven Architecture: Practice and Promise*. Anneke Kleppe, Jos Warmer and Wim Bast. Addison Wesley 2003.
- [Fra03] *Model Driven Architecture. Applying MDA to Enterprise Computing.* David S. Frankel. OMG Press 2003.
- [KY03] *An Evaluation of Compuware OptimalJ Professional Edition as an MDA Tool.* Kings College London & University of York 2003.
- [ORM01] *Architecture Board ORMSC, Model Driven Architecture.* OMG Document ormsc/2001-07-01, 2001.
- [CESW04] *Applied Metamodelling: A Foundation for Language Driven Development.* Tony Clark, Andy Evans, Paul Sammut & James Willans) (available for free at http://www.xactium.com)
- [UML]*The Unified Modeling Language Specification (UML2.0 Superstructure, Final Adopted Specification), ptc/03-08-02*. Contact point: Bran Selic. http://www.omg.org/cgi-bin/doc?ptc/2003-08-02
- [Whi02] *Transformations and Software Modeling Languages*. Jon Whittle. 2002 International Conference on the Unified Modeling Language
- [4+1] *Architectural Blueprints – the "4+1" view model of software architecture.* Phillipe Kruchten. IEEE Software, 12(6), 1995, pps. 42-50