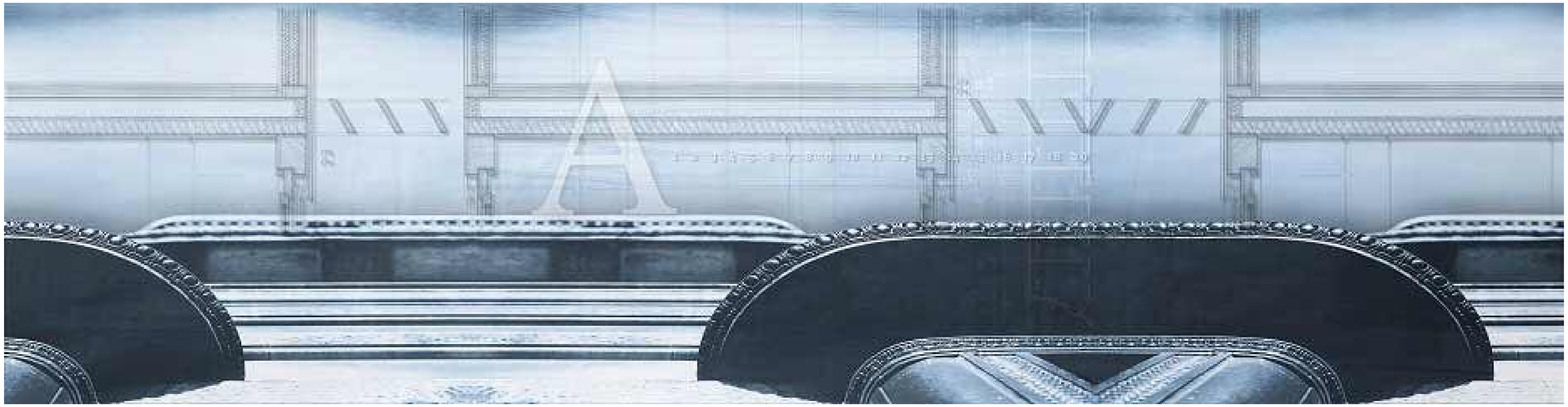


THE IT-ARCHITECTURE PROFESSIONALS

MDA Marks vs. Mapping Chains

Michael Jungmann

www.ArcStyler.com

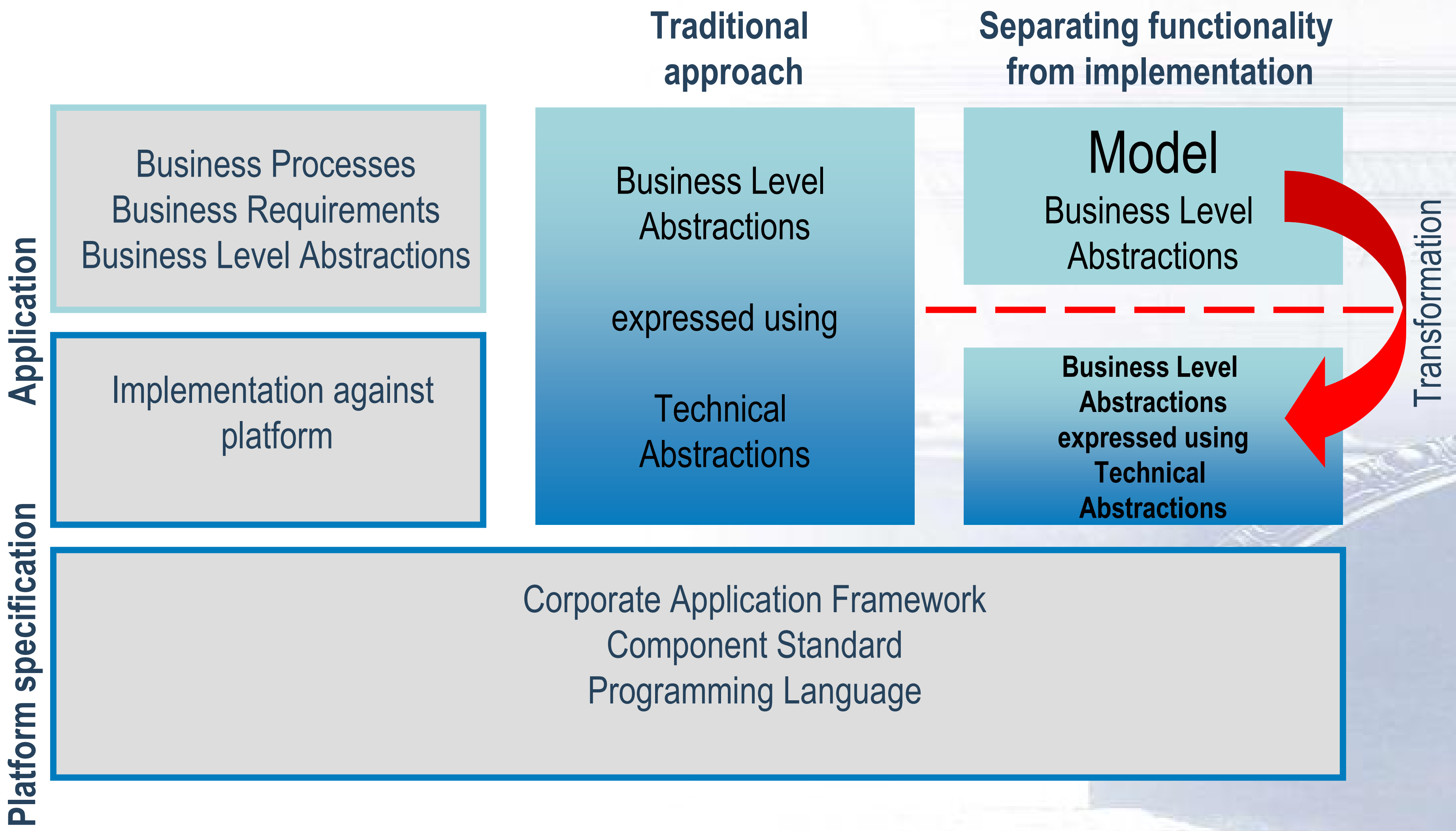
Goal: How to apply MDA successfully

- MDA Overview
  - Big picture
  - What‘s in for me
- MDA core concepts – relations, differences, (suprising) overlaps
  - Mappings
  - Mapping Chains
  - MDA Marks
- MDA challenges and pragmatics – applying MDA successfully
- Things to come

MDA Positioning

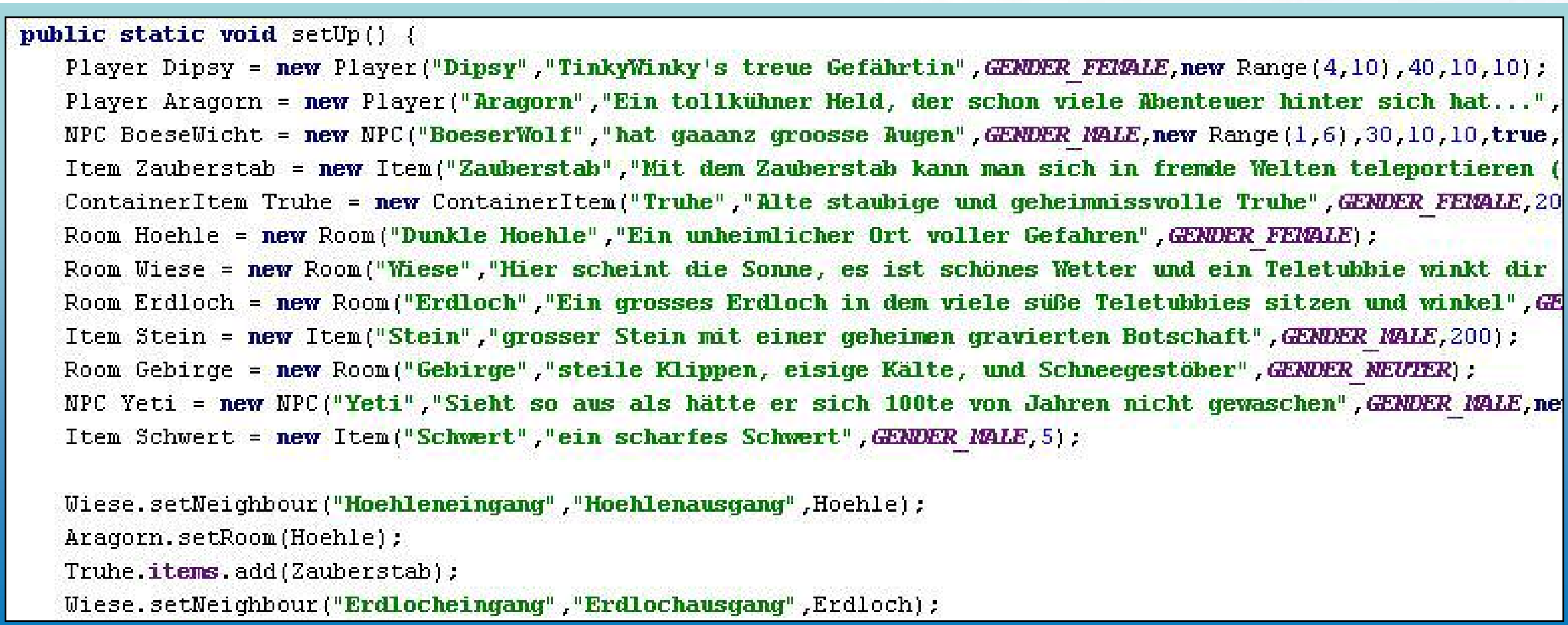
- Standardization initiative of the Object Management Group (OMG)
- Started in 2000 with increasing interest in the industry
- Is aimed at using modeling, meta-modeling and model transformations to drive the design and implementation of distributed systems
- Leverages other OMG standards like
  - Unified Modeling Language (UML)
  - Meta-Object Facility (MOF)
  - XML Metadata Interchange (XMI)
- Allows for system specifications at multiple levels of abstraction
  - Resulting in long-lived, reusable assets instead of just code
- The next consequent step in the evolution of Software Engineering after procedural, object- and component-oriented approaches.

Building Applications


Traditional approach
Separating functionality from implementation
Application
Platform specification
Business Processes
Business Requirements
Business Level Abstractions
Implementation against platform
Business Level Abstractions
expressed using
Technical Abstractions
Model
Business Level Abstractions
Transformation
Business Level Abstractions expressed using Technical Abstractions
Corporate Application Framework
Component Standard
Programming Language

Illustrating Example: Text Adventure Game

- Sample Business: Game Vendor
  - Builds text adventure games on top of application framework

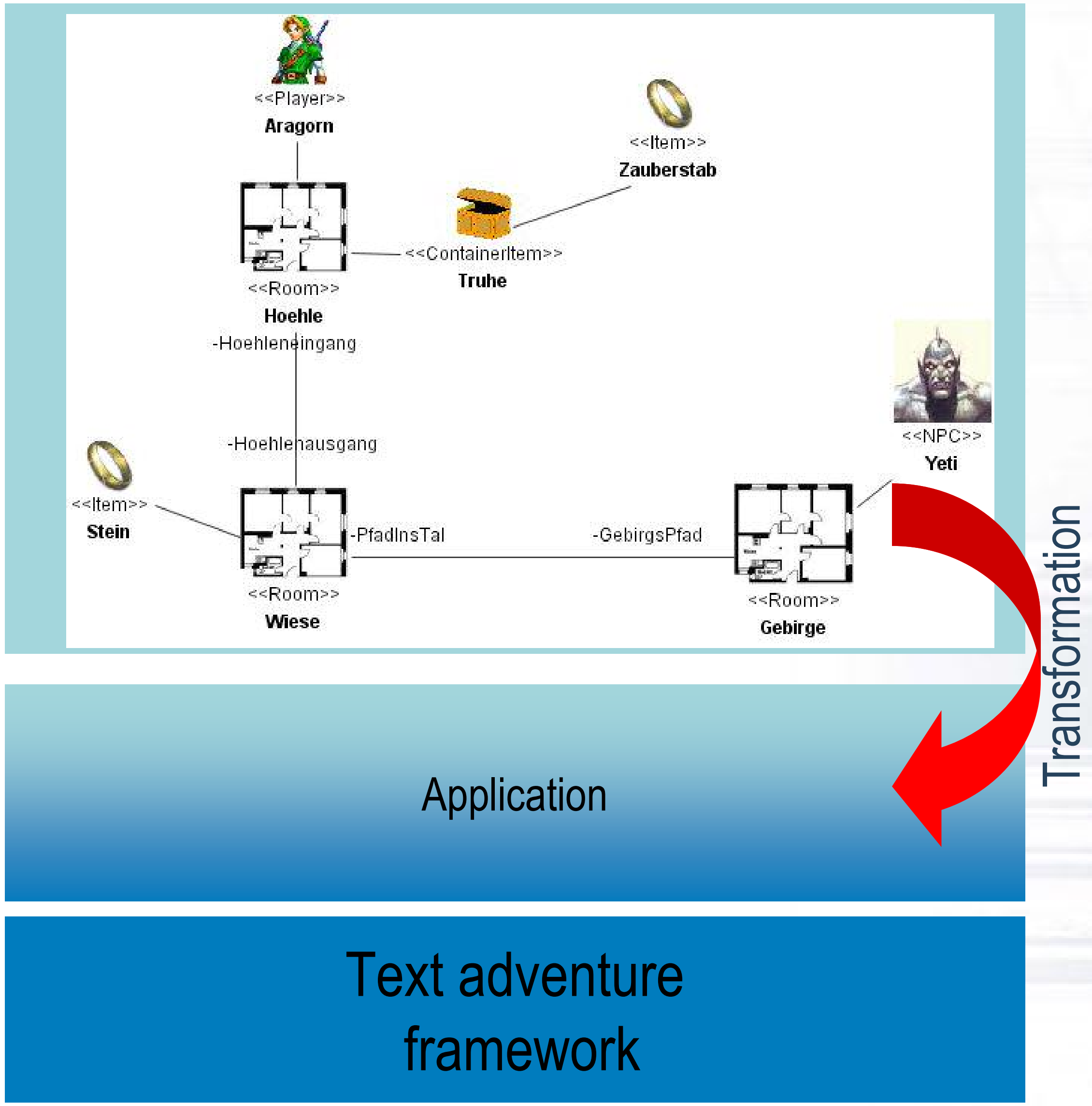


```
public static void setUp() {
    Player Dipsy = new Player("Dipsy","TinkyWinky's treue Gefährtin",GENDER_FEMALE,new Range(4,10),40,10,10);
    Player Aragorn = new Player("Aragorn","Ein tollkühner Held, der schon viele Abenteuer hinter sich hat...",
    NPC BoeseWicht = new NPC("BoeserWolf","hat gaaanz groosse Augen",GENDER_MALE,new Range(1,6),30,10,10,true,
    Item Zauberstab = new Item("Zauberstab","Mit dem Zauberstab kann man sich in fremde Welten teleportieren (
    ContainerItem Truhe = new ContainerItem("Truhe","Alte staubige und geheimnissvolle Truhe",GENDER_FEMALE,20
    Room Hoehle = new Room("Dunkle Hoehle","Ein unheimlicher Ort voller Gefahren",GENDER_FEMALE);
    Room Wiese = new Room("Wiese","Hier scheint die Sonne, es ist schönes Wetter und ein Teletubbie winkt dir
    Room Erdloch = new Room("Erdloch","Ein grosses Erdloch in dem viele süße Teletubbies sitzen und winkel",GE
    Item Stein = new Item("Stein","grosser Stein mit einer geheimen gravierten Botschaft",GENDER_MALE,200);
    Room Gebirge = new Room("Gebirge","steile Klippen, eisige Kälte, und Schneegestöber",GENDER_NEUTER);
    NPC Yeti = new NPC("Yeti","Sieht so aus als hätte er sich 100te von Jahren nicht gewaschen",GENDER_MALE,ne
    Item Schwert = new Item("Schwert","ein scharfes Schwert",GENDER_MALE,5);

    Wiese.setNeighbour("Hoehleneingang","Hoehlenausgang",Hoehle);
    Aragorn.setRoom(Hoehle);
    Truhe.items.add(Zauberstab);
    Wiese.setNeighbour("Erdlocheingang","Erdlochausgang",Erdloch);
```

Text adventure framework

Example ct'd: Applying MDA

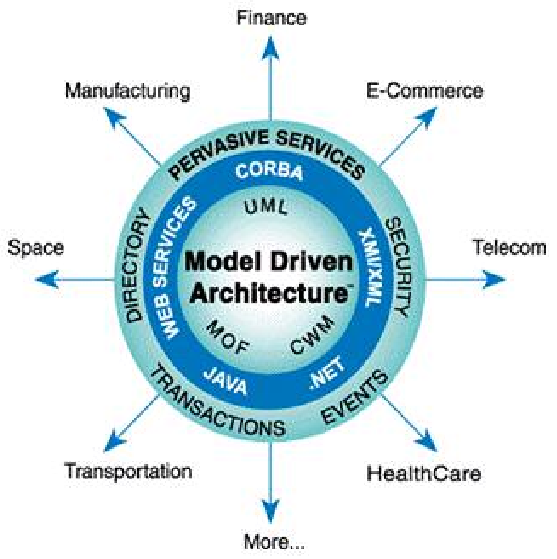


- Model at the business domain level
  - Yes, it is UML
  - Not polluted by technology
  - Easy to understand for domain experts
- Allows for modification at the business domain level without implementation cost
  - E.g. remove a room with all implications
- Allows for retargeting model
  - E.g. creating Web-based applications using a new transformation
  - E.g. creating test scripts

A more simple but more serious example

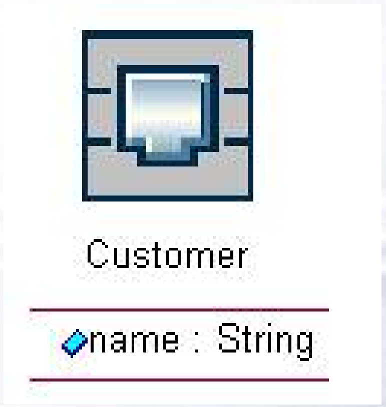
Bean class (EJB 1.1)

```
public java.lang.String name;
```



Modeling

Abstraction



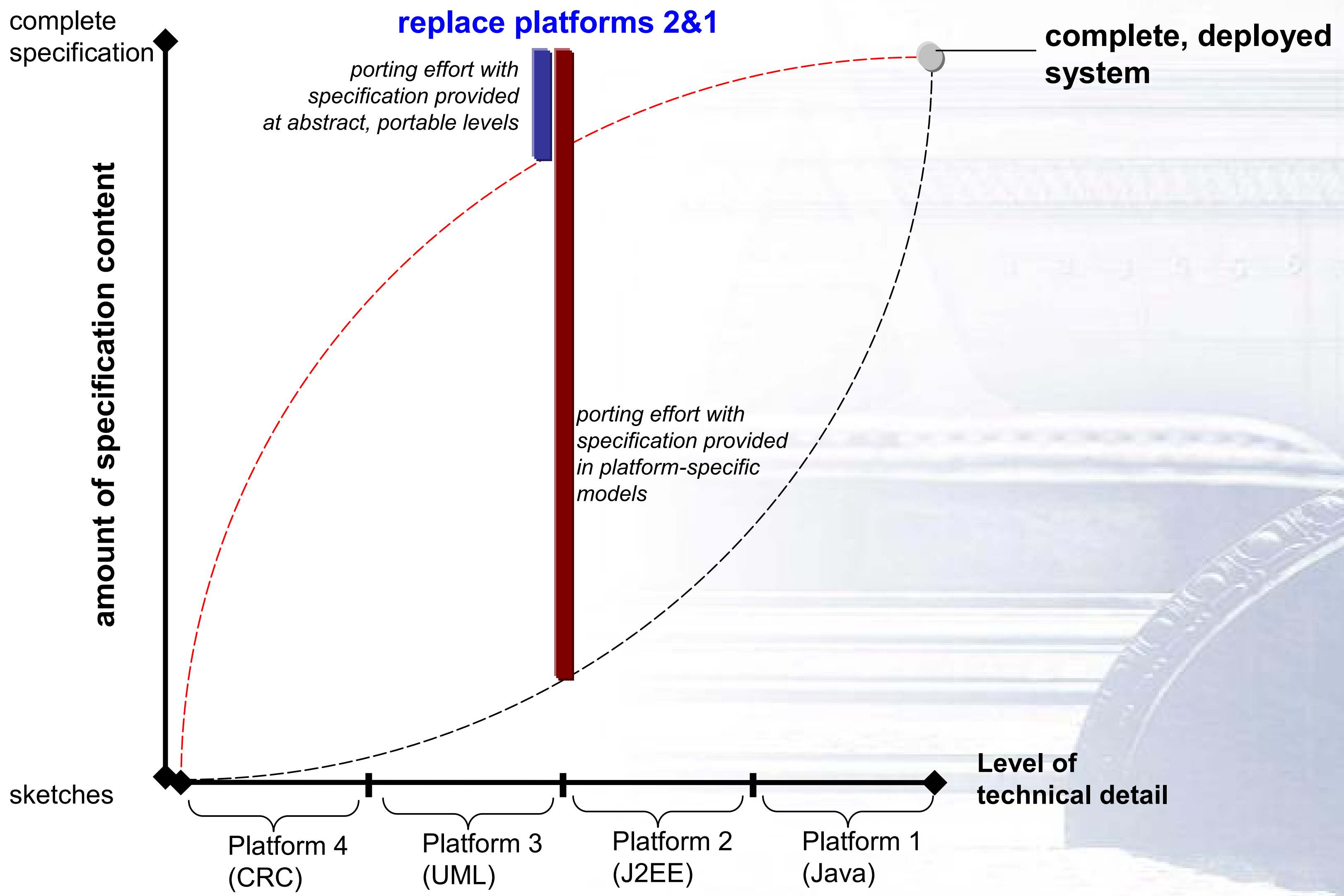
Transformation

Transformation

Bean class (EJB 2.0)

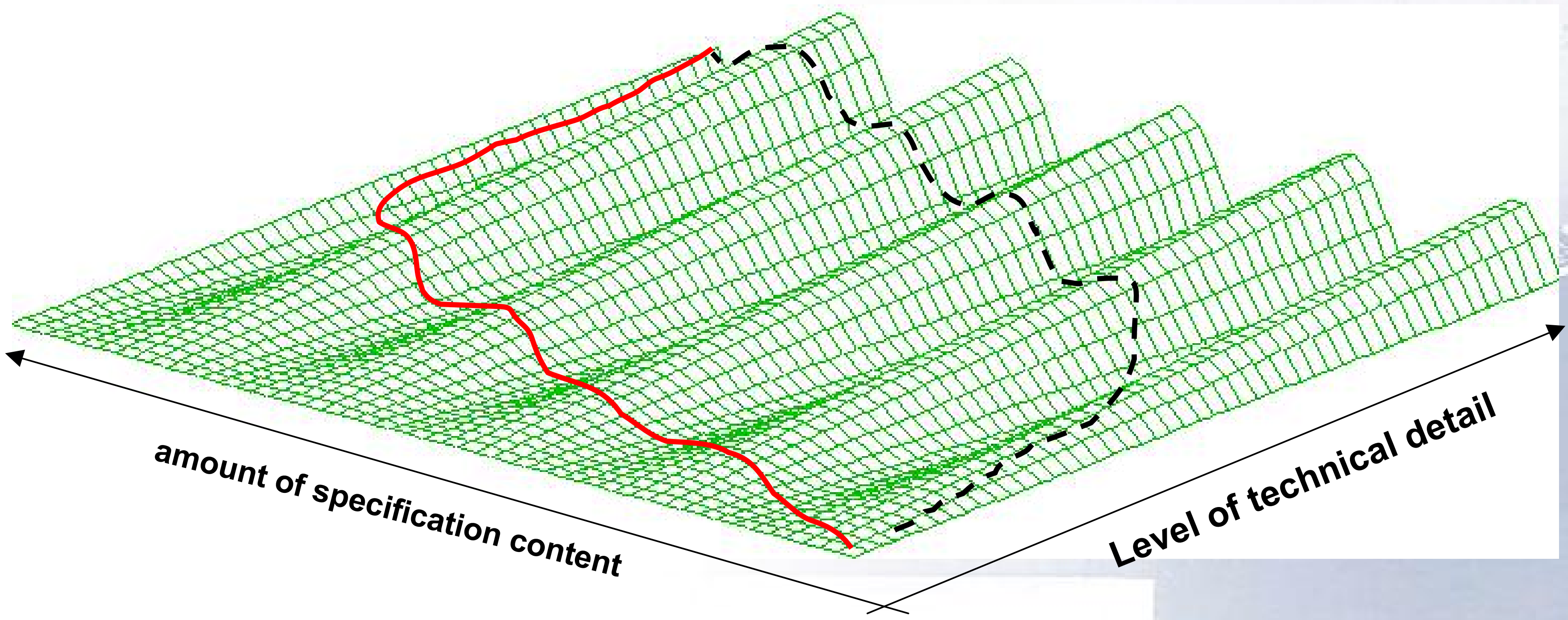
```
public abstract java.lang.String getName();
public abstract void setName(java.lang.String name);
```

MDA Benefits: Improved Portability

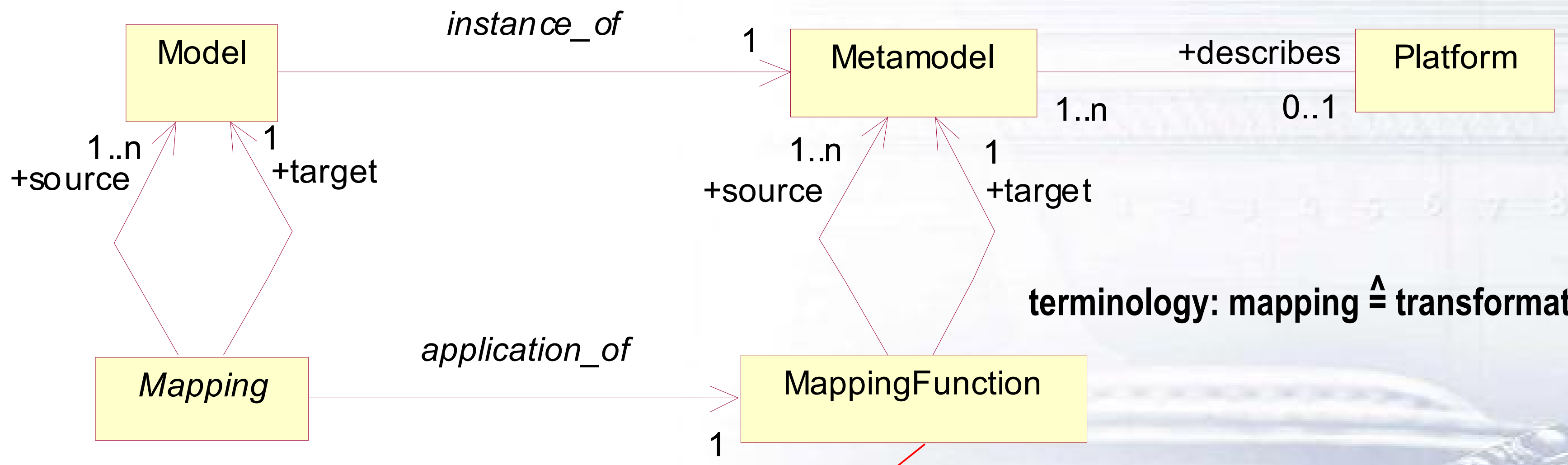

complete specification
replace platforms 2&1
complete, deployed system
porting effort with specification provided at abstract, portable levels
amount of specification content
porting effort with specification provided in platform-specific models
sketches
Level of technical detail
Platform 4 (CRC)
Platform 3 (UML)
Platform 2 (J2EE)
Platform 1 (Java)

MDA Benefits: Increased Productivity

- Strength: Level of abstraction
- Detailing at low abstraction level causes extra effort and errors.
- Example: Associations between EJB components



The Heart of MDA: Transformations



terminology: mapping ≙ transformation

Specification / implementation getting standardized by MOF 2.0 Query / View / Transformation (QVT) RFP. Implementation also called "*Cartridge*".

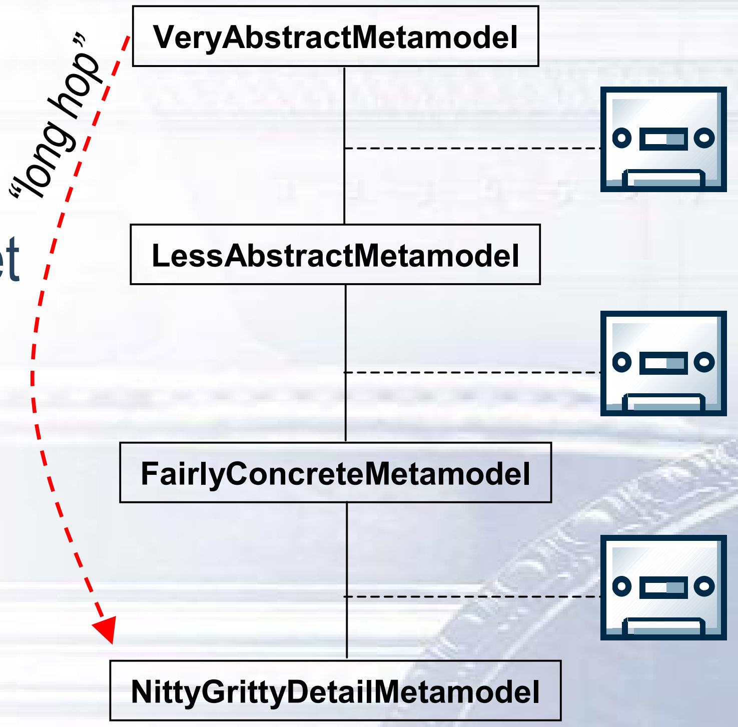
Multiple mappings may be applied successively in a ***chain***

Mapping Function Implementation

- MOF as basis for metamodels becomes common
- Exception: ASCII-based target "metamodels"
- Specification of mapping functions varies
  - ASCII-based target metamodels
    - templates mixed with scripting (akin to JSP approach)
    - UML-based (ArcStyler / CARAT)
    - XML / XSLT
    - hard-wired
    - mostly irreversible (often not possibly without ambiguities)
  - MOF-based target metamodels
    - rules-based
    - hard-wired
    - sometimes reversible (when unambiguously possible)

Chaining Mapping Functions

- Benefits
  - Allow modelers to use appropriate abstractions at all times
  - Increase reusability of mapping functions and models
  - Allow for manual refinement of target model
- Risks
  - Longer turnaround cycles than with long hop transformation
  - Versioning issues
  - Difficulties in maintaining manual refinements in target models

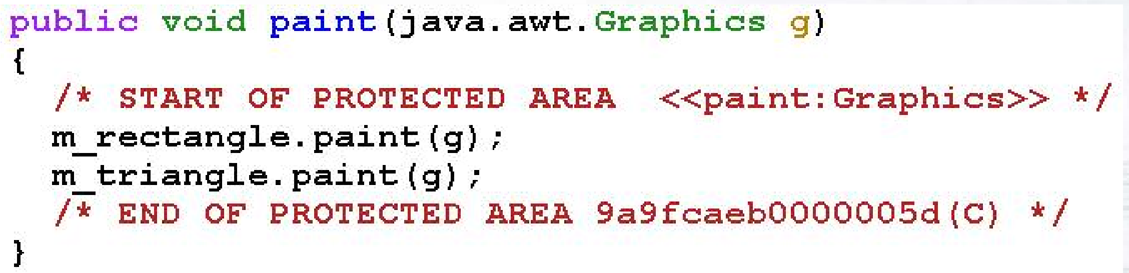


Handling Manual Changes in Target Models

- Manual changes to target models can represent valid refinement / elaboration
- Changes should be preserved “as good as possible” upon repeatedly applying the mapping function
- Example
  - mapping function produces skeletal code
  - operation bodies are filled in manually
  - adding an attribute in the model requires re-generation
  - preserve manually-edited operation bodies, add attribute implementation (get-/set-operation, attribute member)

Solution Approach for Text-Based “Models”

- Protected areas
  - mark-up in code
  - hash function defining the notion of “manually changed”
  - store hash-code in mark-up
  - in case of manual change, preserve area contents
- Protected areas with merge logic
  - as above, but in case of manual change:
  - merge new default content with manual changes



```
public void paint(java.awt.Graphics g)
{
  /* START OF PROTECTED AREA  <<paint:Graphics>> */
  m_rectangle.paint(g);
  m_triangle.paint(g);
  /* END OF PROTECTED AREA 9a9fcaeb0000005d(C) */
}
```

Model-to-Model Mapping Functions

- Challenge #1: Define kinds of preservable changes
- Examples of manual changes in a UML model:
  - change the name of a class
  - add an attribute
  - remove an attribute
  - change ordering of attributes
- Challenge #2: Define synchronization rules
- Examples for synchronization options:
  - leave list of attributes as is
  - don’t re-add deleted attributes; add new attributes

MDA Marks

- So, this was about mapping functions
- … and chances and risks when chaining them

- … now, let‘s turn over to MDA Marks

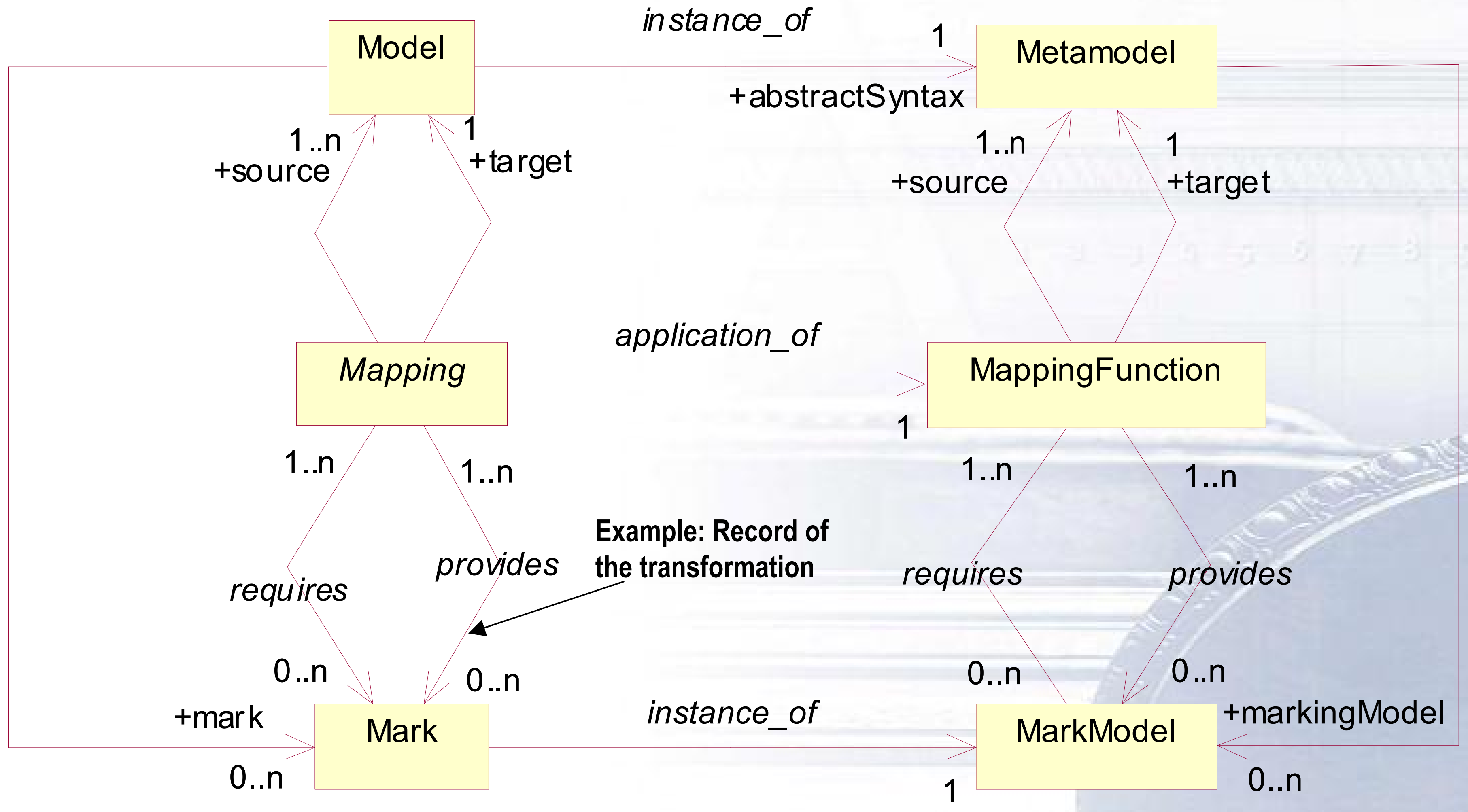
Marking Up Models

- Keep models as portable as possible!
  - i.e. independent of mapping function and target metamodel
- Add mapping-specific information
  - Parameters to the transformation per model element
- Solution: MDA Marks
- *Marks* are instances of *marking models* which:
  - Are comparable to Post-Its
  - Refine the model without “polluting” it
  - Work together with mapping functions
  - Describe their structure and semantics
- In UML, often *implemented* using Tagged Values

Customer
I want
To become
An Entity Bean

Marks for Specific Mapping Functions

The MDA "***Six-Box***" Model

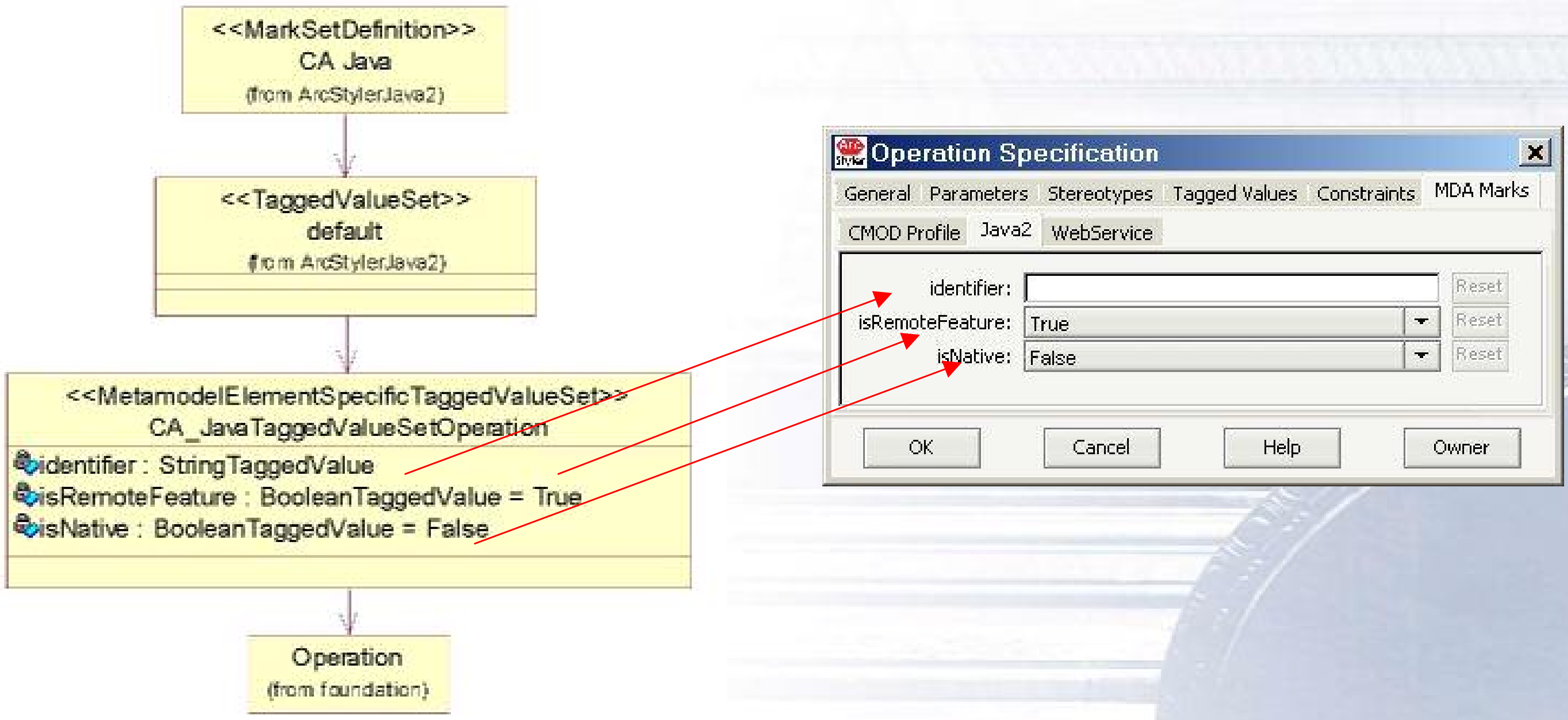


Example: Record of the transformation

Example: Marking Model and Marks

marking model for meta-class ***Operation***

editing the resulting mark instances



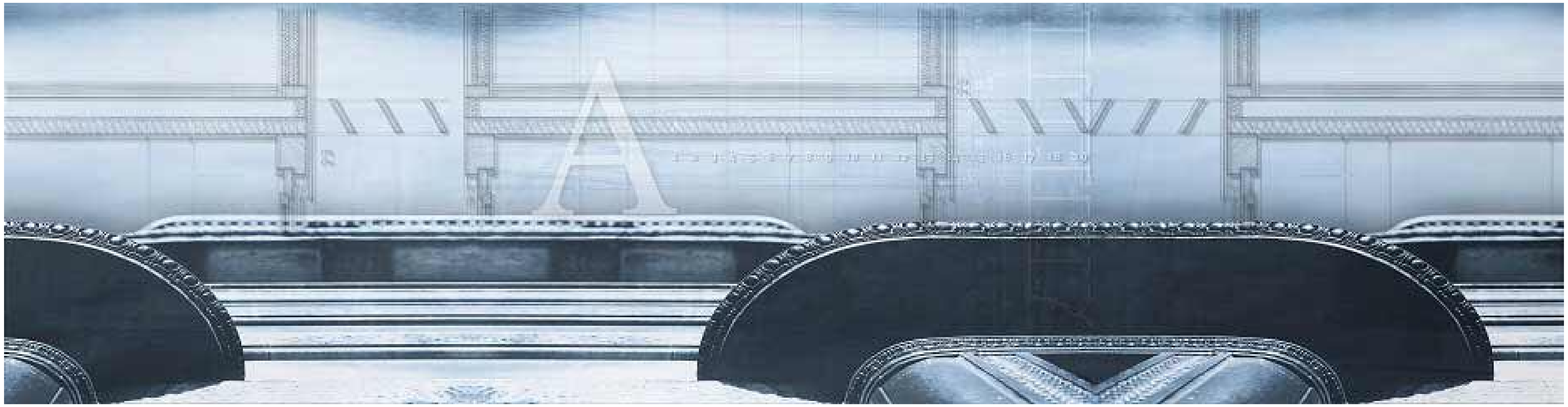
identifier:
isRemoteFeature: True
isNative: False

Marks vs. Mapping Chains

- Marks and Mapping Chains are really different concepts
  - Marks are additional parameters to a transformation and are essential to MDA
  - Mapping chains are a means for composing transformations and refining intermediate models
- **Overlap occurs, where Marks are used for representing refinement information**
  - e.g., *marking* a UML attribute as being persistent – this could be done either using marks or in the target model
- **In practice, there is often the choice between applying mapping chains and marks for the purpose of refining the target model**
  - Trade power and complexity/cost of mapping chains against simplicity and robustness of long-hop transformations using marks
  - As appropriate in a given situation

Heading for MOF 2.0 QVT

- First prototypical implementations available
  - ArcStyler
  - TRL
  - OptimalJ
- Work in progress
  - iterative mapping function execution
  - identifying and preserving manual changes
  - specify change idenfication and preservation in mapping rules
- As MDA matures, mapping chains will become less complex and easier to handle
- Until then, applying marks in favor of mapping chains where appropriate reduces risk and cost in real-world MDA projects



THE IT-ARCHITECTURE PROFESSIONALS

Model Driven Architecture
for the Enterprise

http://www.ArcStyler.com

Interactive Objects Software GmbH
Basler Strasse 65
79100 Freiburg, Germany

Tel. [+49] 761 / 4 00 73 - 0
Fax [+49] 761 / 4 00 73 – 73

info@iO-Software.com