# Software Industrialization

## A Perspective on MDA®

**David Frankel Consulting**
*df@DavidFrankelConsulting.com*
*www.DavidFrankelConsulting.com*

**Portions adapted from the book**
**Model Driven Architecture: Applying MDA to Enterprise Computing**
**David S. Frankel**

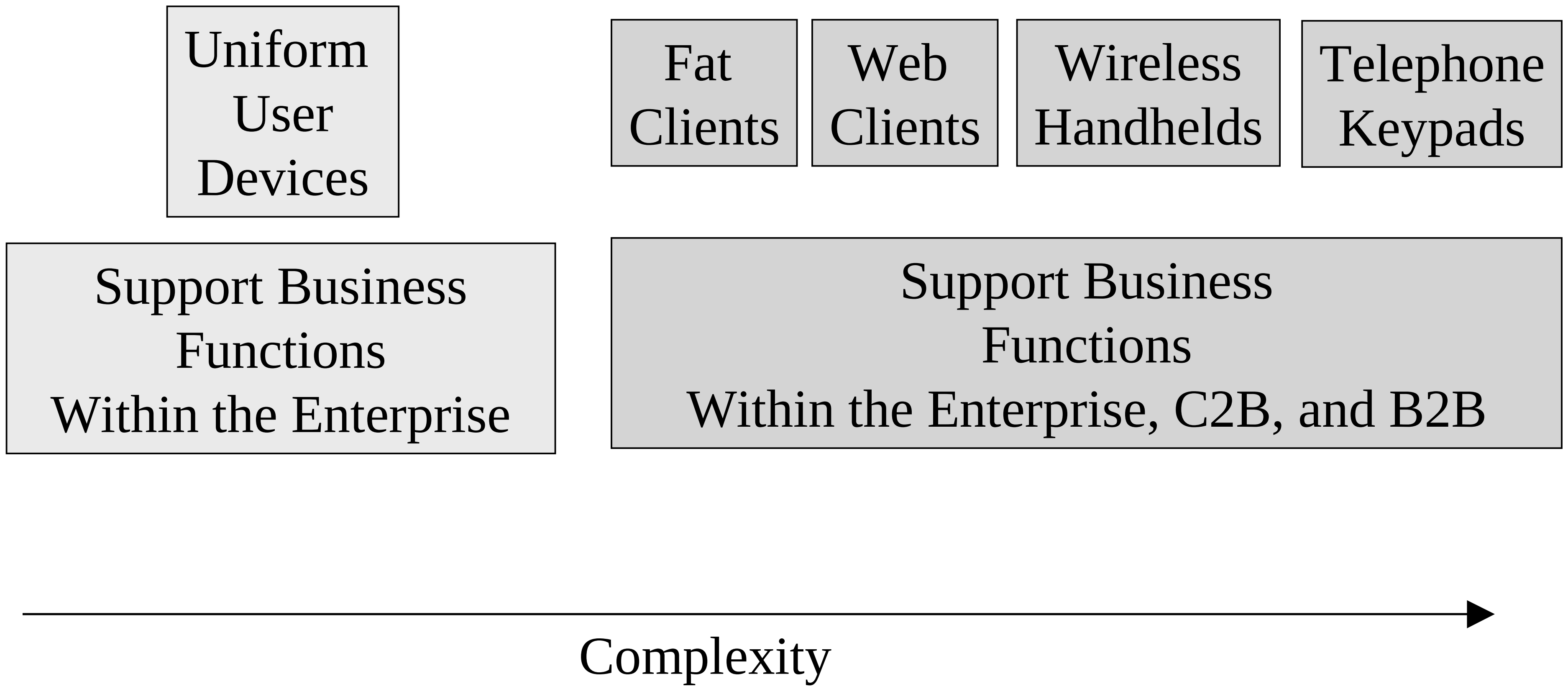
# Agenda

- The Demands of the Virtual Enterprise
- MDA: Industrializing Software
- Informal vs. Formal Modeling
- Future MDA Directions

## Agenda

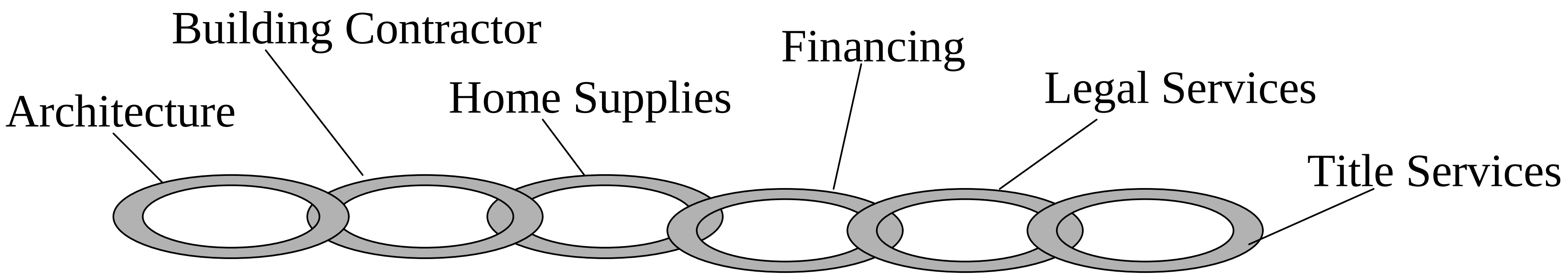
- The Demands of the Virtual Enterprise
- MDA: Industrializing Software
- Informal vs. Formal Modeling
- Future MDA Directions

# Increased Complexity Facing IT


Uniform User Devices
Support Business Functions Within the Enterprise
Fat Clients
Web Clients
Wireless Handhelds
Telephone Keypads
Support Business Functions Within the Enterprise, C2B, and B2B
Complexity

# Value Chain Driven Business

## Rapid Assembly of Value Chains



## Issues

- Building, updating, and integrating these complex distributed systems is labor-intensive
  - Easy to use a good application server in an unscalable fashion
- Many projects fail
  - Others have pointed this out

## Agenda

- The Demands of the Virtual Enterprise
- ➡ MDA: Industrializing Software
- Informal vs. Formal Modeling
- Future MDA Directions

## MDA and Industrialization

- Accelerates the trend toward automating low-level programming
- Applies principles of industrial manufacturing to achieve efficiencies and automation
  - Formal blueprints
  - Components
  - Patterns
- Crawl, walk, run…a gradual change

## Bringing Model-Centrism to Intermediate Tiers, EAI and B2Bi

- Part of general trend to raise the abstraction level
- Models as development artifacts
  - Not simply blueprints for humans
- Already well-established for front and back ends
  - WYSIWYG GUI modeling and data modeling
  - Hand coding no longer predominates
  - But tuning allowed
- Wizards vs. models

## Component-Based Development

- Interchangeable components and scientific management were the keys to the industrial revolution
- More than objects: Independently deployable
- Excellent source: *Business Component Factory*, by Peter Herzum and Oliver Sims
- Service Oriented Architecture
  - Driven by value chain imperative

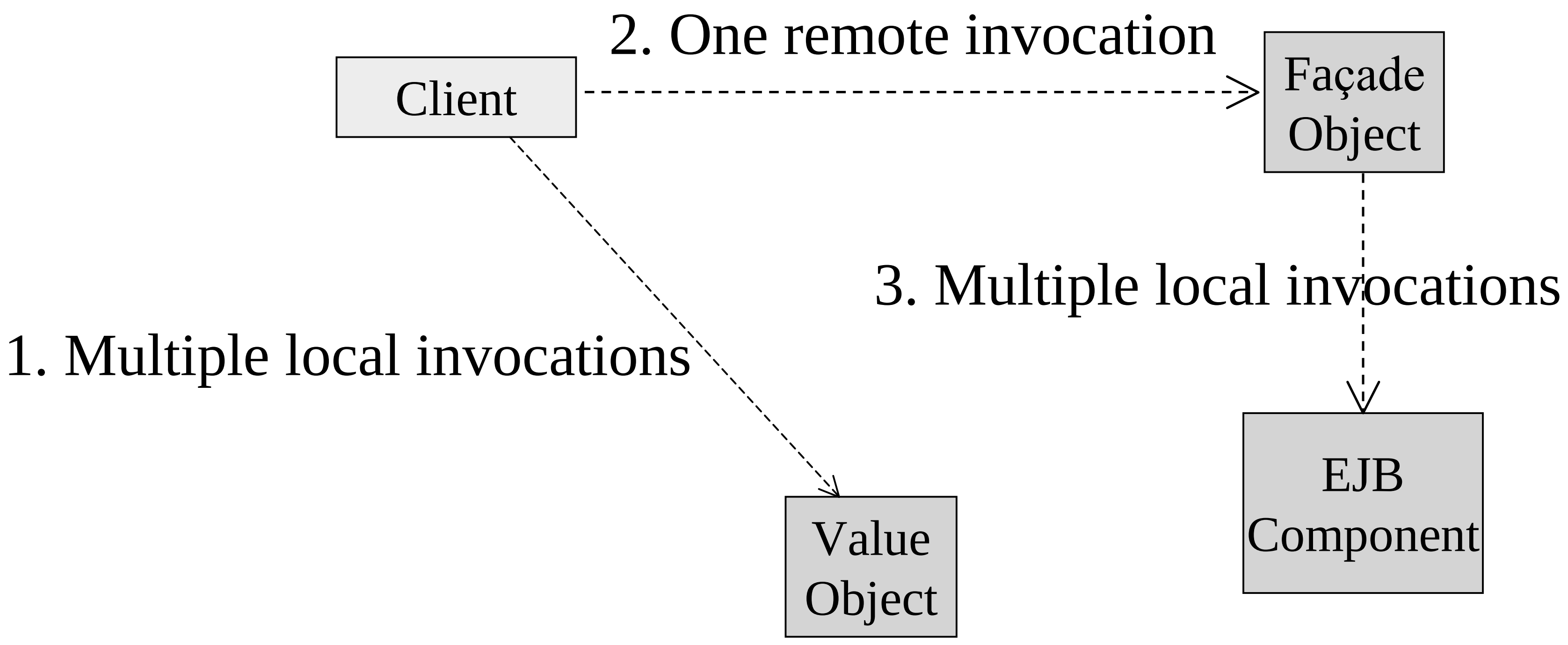
## Design Patterns

- Patterns at the technical level
  - Such as *Java Blueprints*
  - Best practices for implementing components or a set of interacting components
- Some patterns make sense at the level of business semantics
  - Such as the *Observer* pattern (Gamma et al)

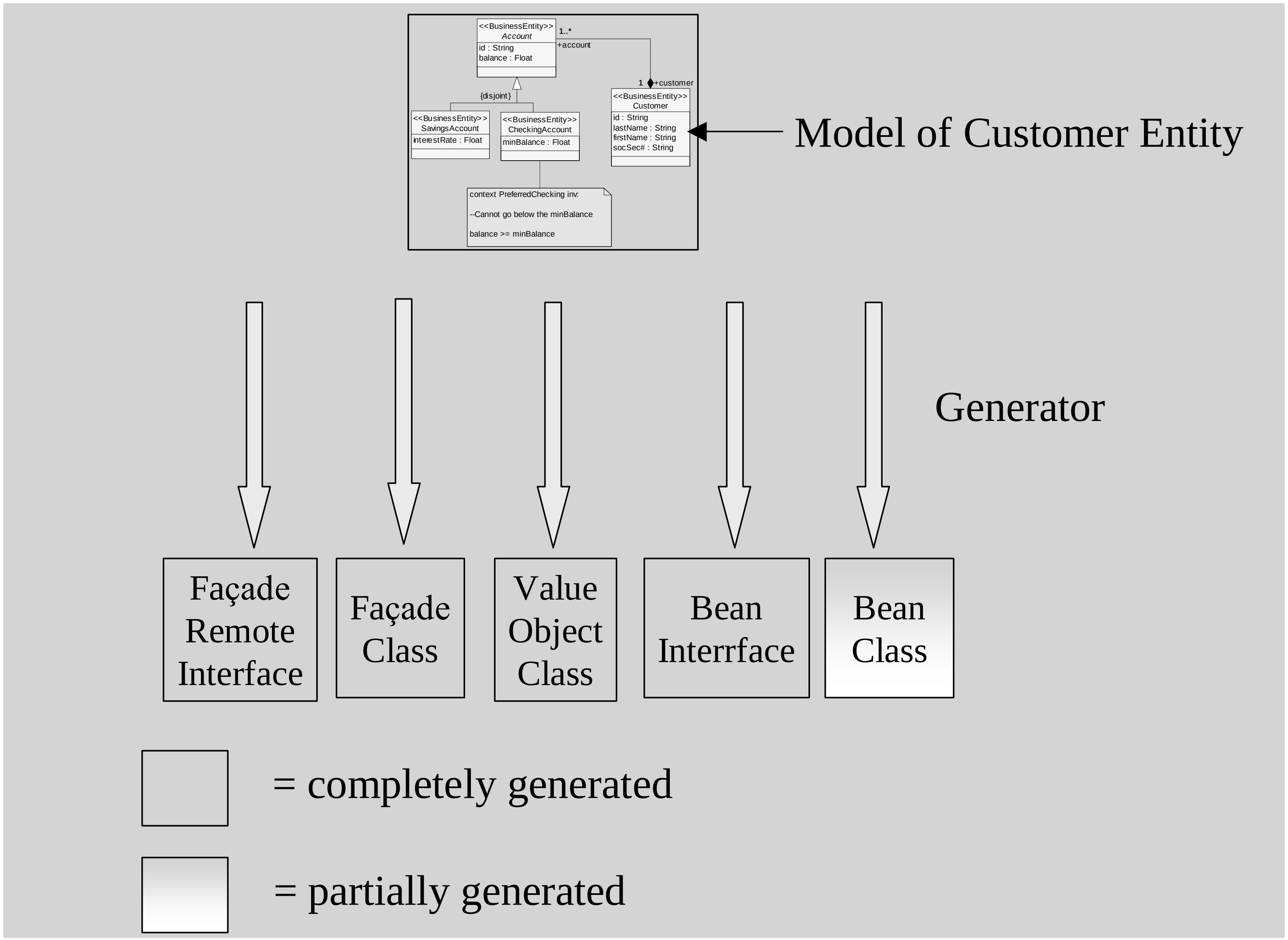
## Automatic Pattern Replication

- MDA generators encapsulate pattern knowledge
  - And apply patterns automatically
  - Technical patterns are the most amenable
  - Repetitive hand-coding of each pattern instance is inefficient
  - Patterns community is coming around to this view
    - e.g. John Crupi
- Generators can enforce large scale patterns or *architectural styles*
  - Richard Hubert, *Convergent Architecture*

# Using Value Object Design Pattern to Set Attributes



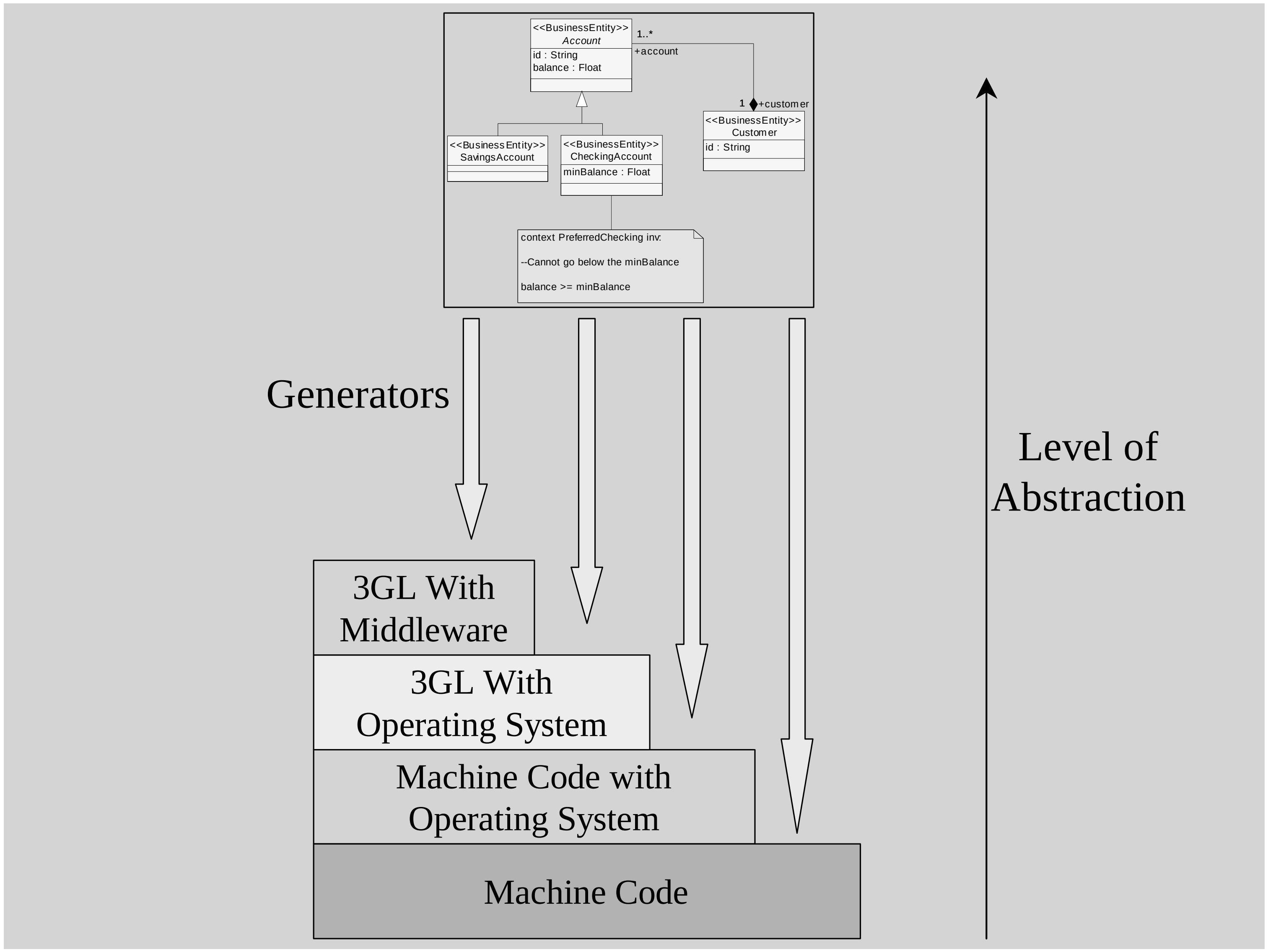
# A Generator Applying the Value Object Pattern



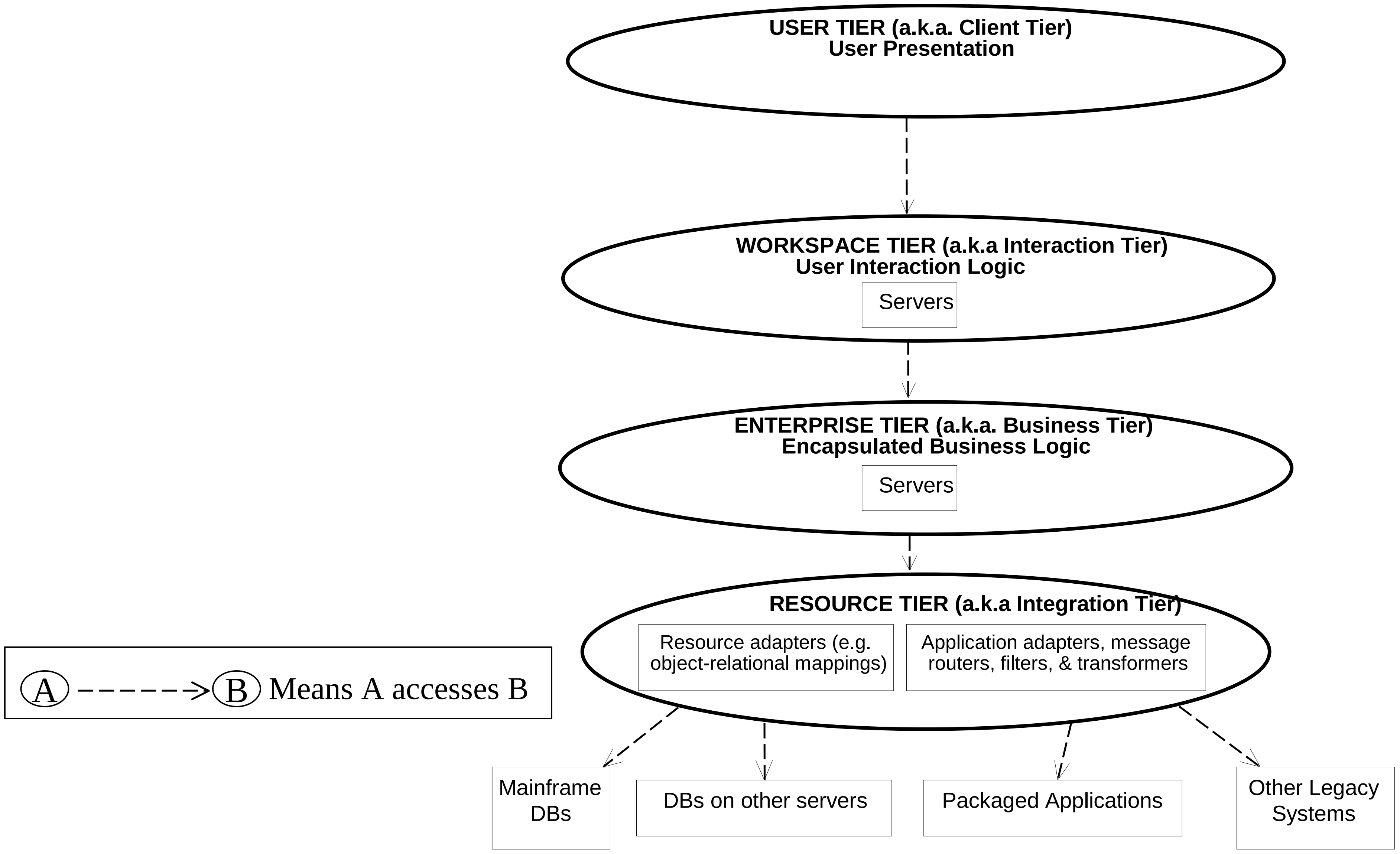
## Other Advances Toward Efficiency

- Middleware
  - Raises the abstraction level of the platform
- Declarative Specification
  - e.g. setting transaction properties in component descriptors
- Enterprise Architecture
  - Separation of concerns

# Middleware Narrows the Abstraction Gap

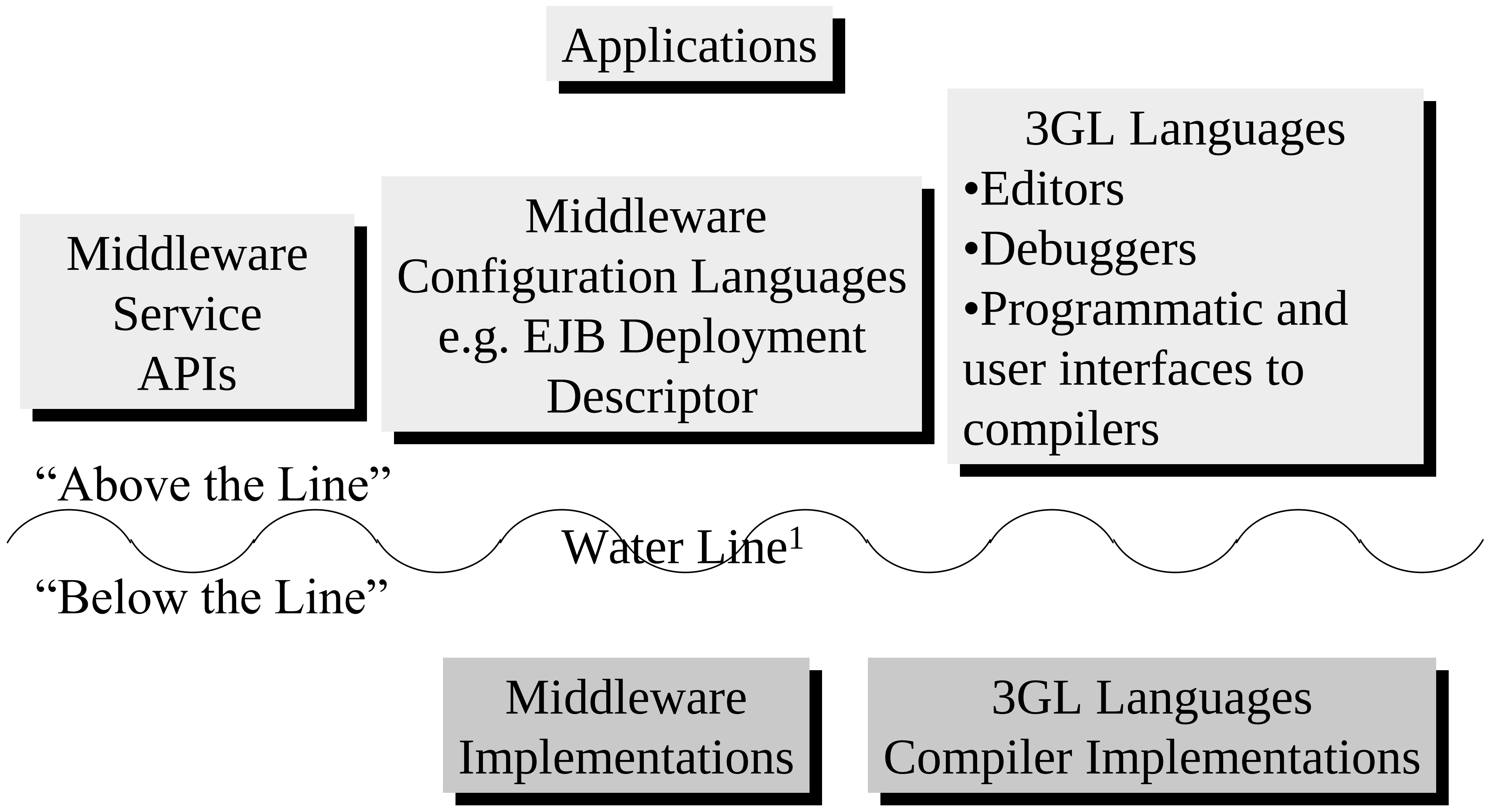


# Multi-Tiered Enterprise Architecture With EAI Adapters & Message Management



USER TIER (a.k.a. Client Tier)
User Presentation
WORKSPACE TIER (a.k.a Interaction Tier)
User Interaction Logic
Servers
ENTERPRISE TIER (a.k.a. Business Tier)
Encapsulated Business Logic
Servers
RESOURCE TIER (a.k.a Integration Tier)
Resource adapters (e.g. object-relational mappings)
Application adapters, message routers, filters, & transformers
Mainframe DBs
DBs on other servers
Packaged Applications
Other Legacy Systems
A
B
Means A accesses B

# Architectural Separation
## Application Viewpoint vs. Infrastructure


Applications
Middleware Service APIs
Middleware Configuration Languages e.g. EJB Deployment Descriptor
3GL Languages
•Editors
•Debuggers
•Programmatic and user interfaces to compilers
"Above the Line"
Water Line[1]
"Below the Line"
Middleware Implementations
3GL Languages Compiler Implementations

[1]The "above and below the line" concept was developed by Oliver Sims

## Model-Driven Enterprise Architecture

- UML “out of the box” does not support modeling enterprise-centric computing
  - Tiers
  - Middleware layers
  - Distributed components
  - Security
- A model-driven enterprise architecture requires modeling languages to support it
  - Distinct but coordinated
  - For different system aspects and levels of abstraction
  - Use UML profiles and MOF to define the languages

# MDA Architectural Resources

## Above and Below the Line

Applications

Modeling Languages

- Editors (e.g. UML modeling tools)
- Programmatic and user interfaces to generators

Water Line

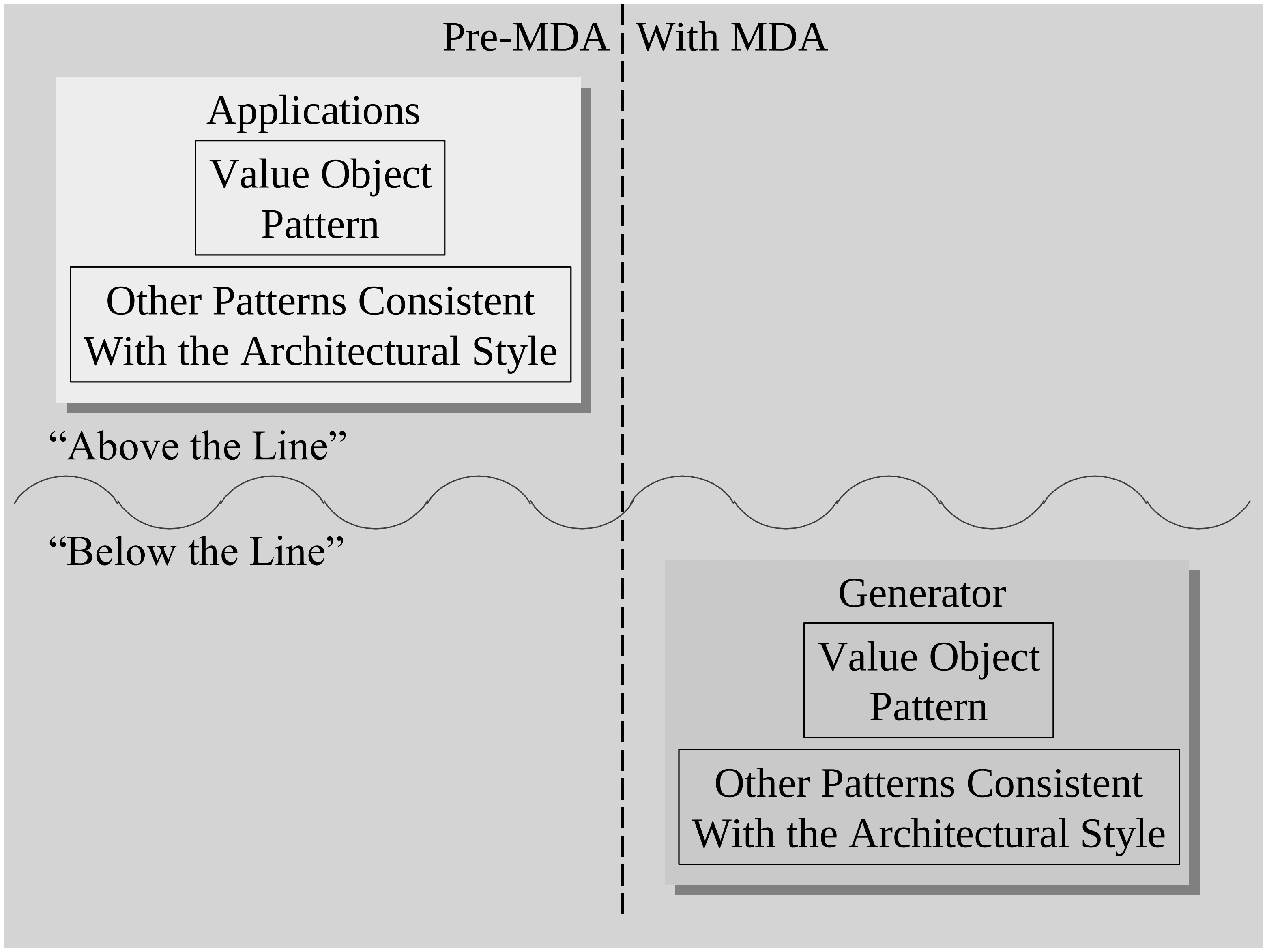
Modeling Language Definitions (language creator's Viewpoint)

Mappings of languages to Technologies (including application of patterns)

Generator Implementations (Generators conform to mappings)

* = At least partially standardized

# Pushing Pattern Knowledge Below the Line


Pre-MDA
With MDA
Applications
Value Object Pattern
Other Patterns Consistent With the Architectural Style
"Above the Line"
"Below the Line"
Generator
Value Object Pattern
Other Patterns Consistent With the Architectural Style

## Model-Driven Development vs. Model Driven Architecture

- MDA includes model-driven development
- Also about model-driven deployment
  - Currently deployment tools metadata is fragmented
    - Little standardization
- Also about model-driven management (ops)
  - Generating instrumentation from models of service-level agreements (SLAs)
  - Java Management Specification (JSR-77) provides some standardization

## Agenda

- ➡ The Demands of the Virtual Enterprise
- MDA: Industrializing Software
- Informal vs. Formal Modeling
- Future MDA Directions

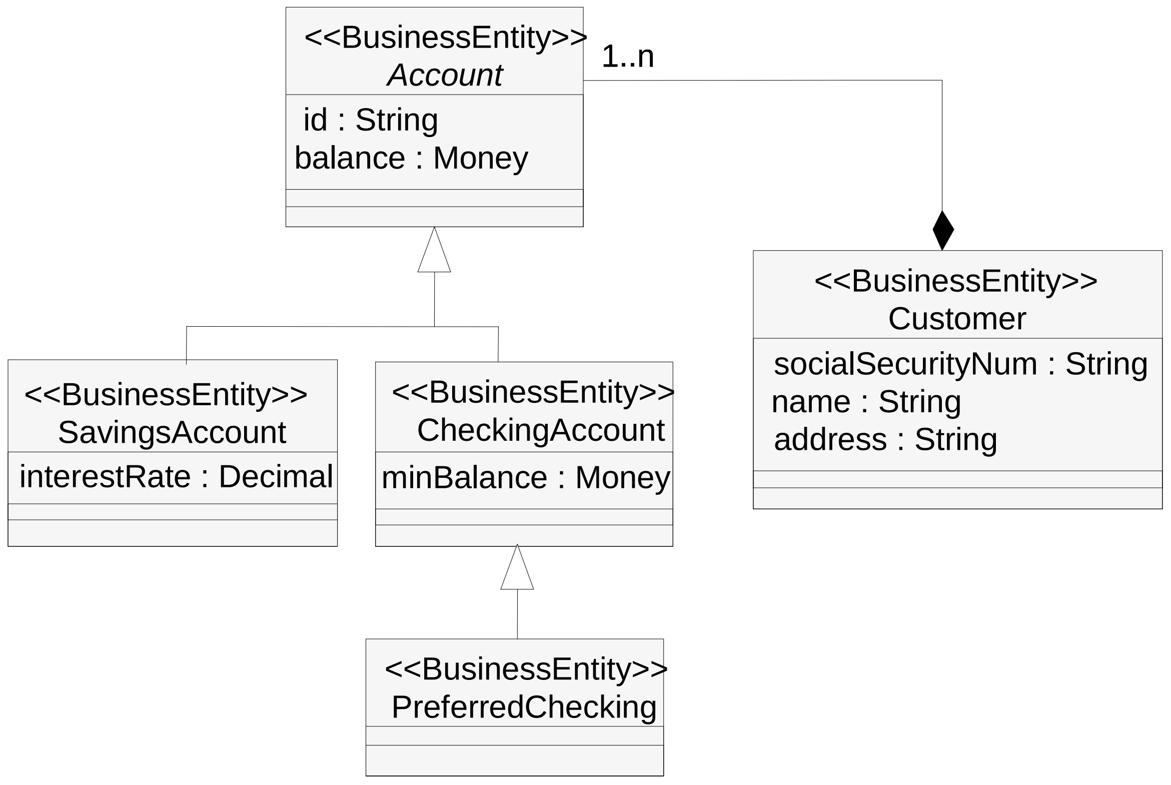
## Informal Models

- Informal modeling
- Used to sketch out basic concepts
- Advantage over typical box and line diagrams because shapes and line types have specific meanings
- Important, but can’t drive code generators and dynamic execution engines
  – Analogously, informal text can’t be compiled and executed like 3GL text

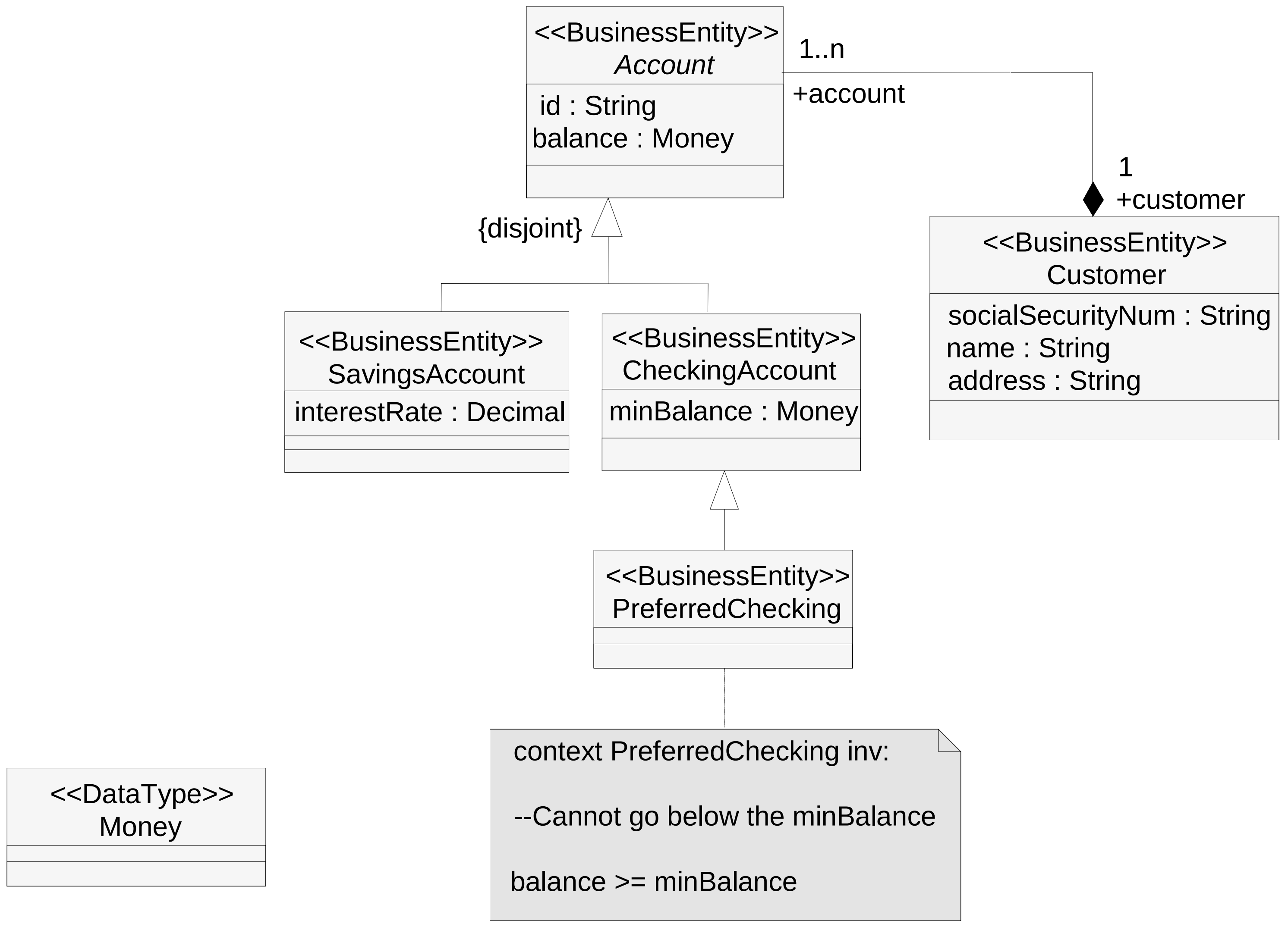
# Formal Models

- Precise
  - Precision and detail are *not* the same!
- Computationally complete
  - Missing properties and unresolved references not acceptable
  - 3GL analogy…
    - an incomplete expression such as “a +” does not compile
    - An undeclared identifier does not compile

# Business Information Model
## Imprecise and Incomplete



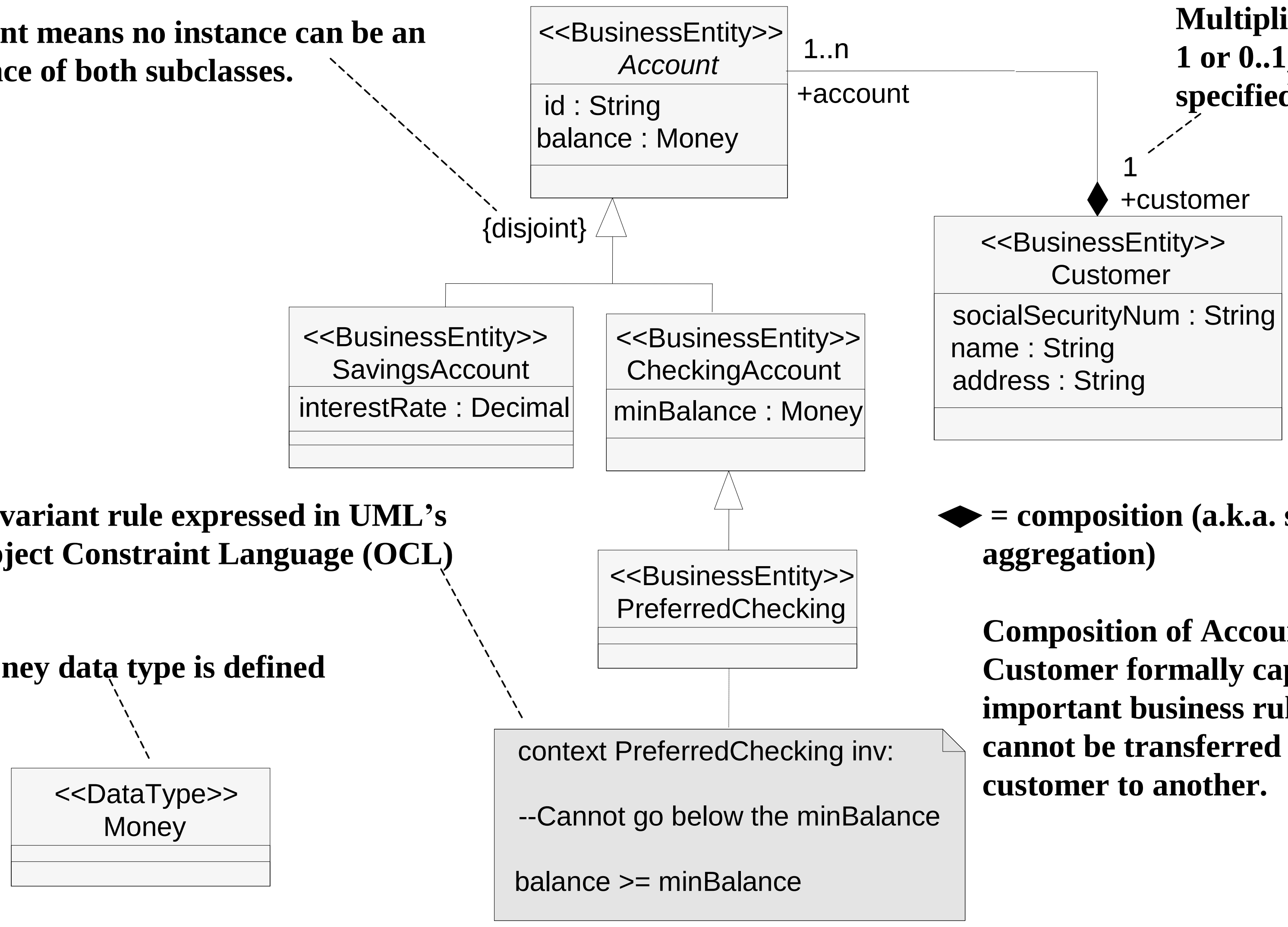
# Business Information Model
## Precise and Complete



# Business Information Model
## Precise and Complete

**Disjoint means no instance can be an instance of both subclasses.**

**Multiplicity could be 1 or 0..1, must be specified**



```
context PreferredChecking inv:

--Cannot go below the minBalance

balance >= minBalance
```

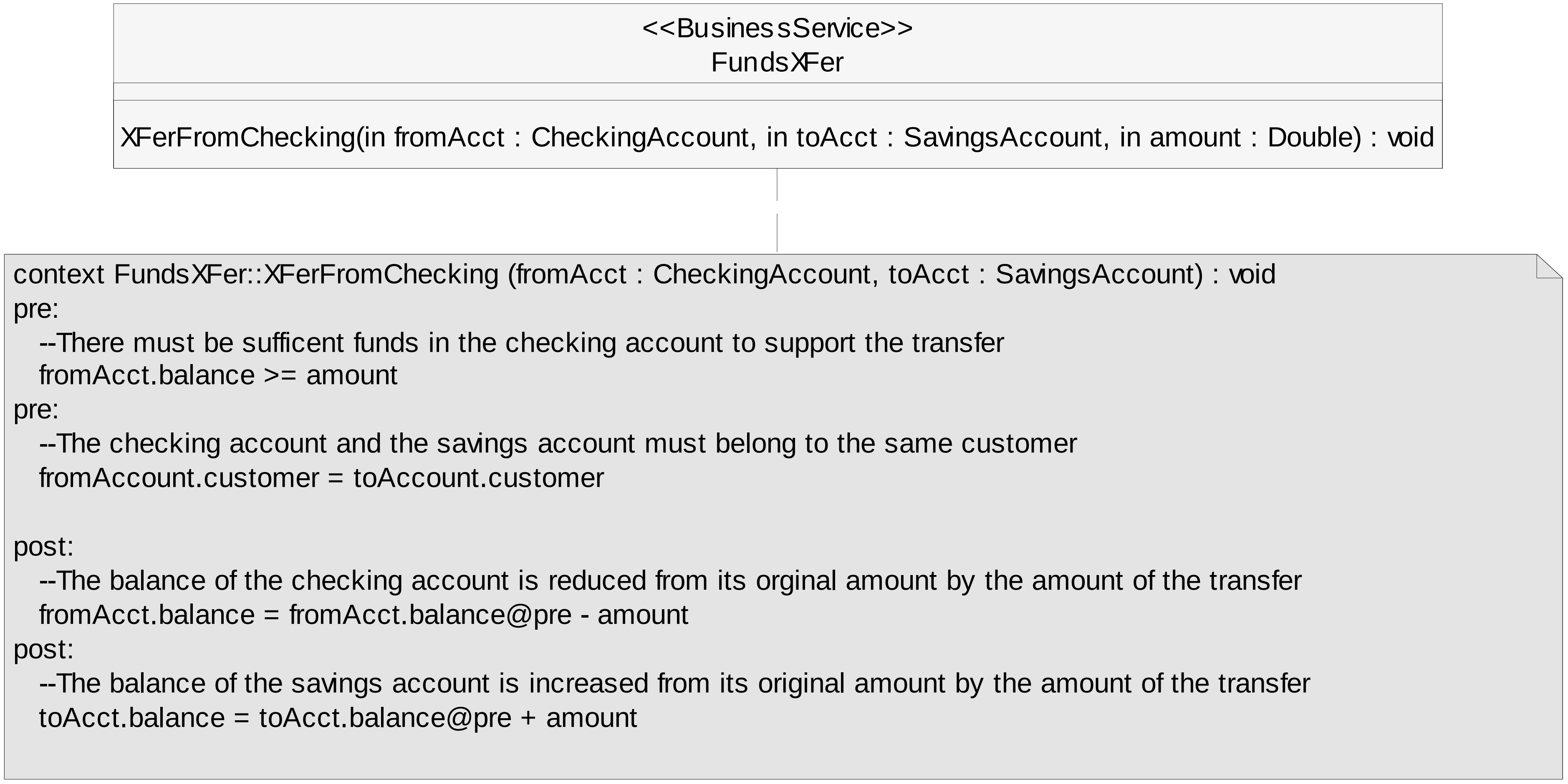
**Invariant rule expressed in UML's Object Constraint Language (OCL)**

**Money data type is defined**

**◆ = composition (a.k.a. strong aggregation)**

**Composition of Account by Customer formally captures an important business rule: An account cannot be transferred from one customer to another.**

# A Formal Model of an Abstract Business Service


<<BusinessService>>
FundsXFer
XFerFromChecking(in fromAcct : CheckingAccount, in toAcct : SavingsAccount, in amount : Double) : void

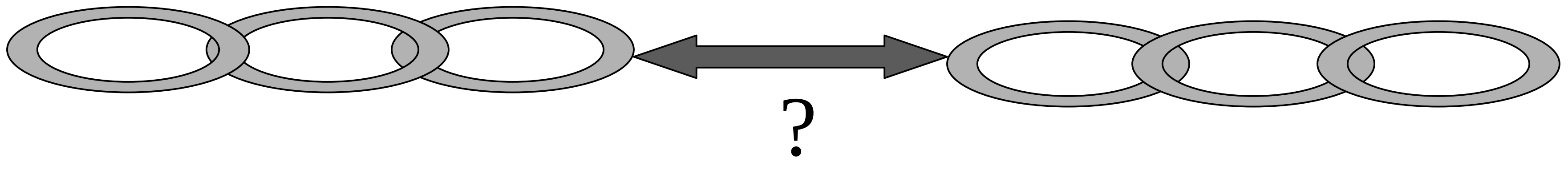
```
context FundsXFer::XFerFromChecking (fromAcct : CheckingAccount, toAcct : SavingsAccount) : void
pre:
   --There must be sufficent funds in the checking account to support the transfer
   fromAcct.balance >= amount
pre:
   --The checking account and the savings account must belong to the same customer
   fromAccount.customer = toAccount.customer

post:
   --The balance of the checking account is reduced from its orginal amount by the amount of the transfer
   fromAcct.balance = fromAcct.balance@pre - amount
post:
   --The balance of the savings account is increased from its original amount by the amount of the transfer
   toAcct.balance = toAcct.balance@pre + amount
```

## Contracts, Reuse, and Interoperability

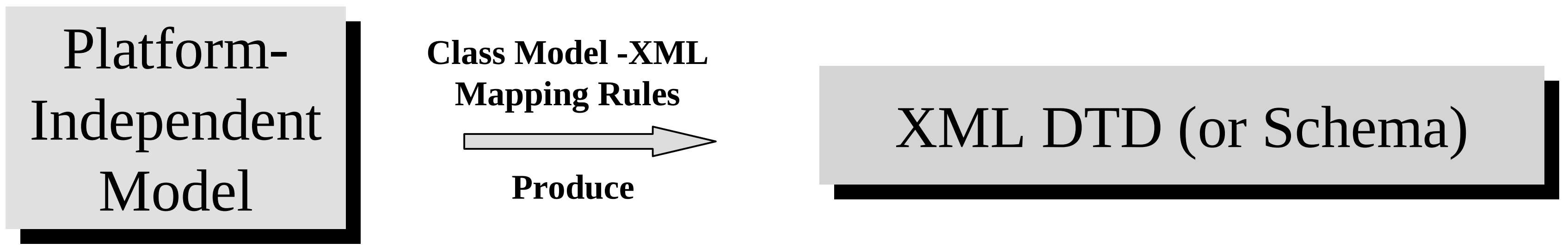
- "Connecting the dots"
  - Makes the specification more complete
  - Flushes out design flaws
- Interoperability among components is difficult when contract not well understood
- Formal contract increases the degree of semantic interoperability
  - Regardless of whether code is generated from the contract
  - Semantic interoperability required for B2Bi
- Provides a "gold standard" for people who speak different human languages

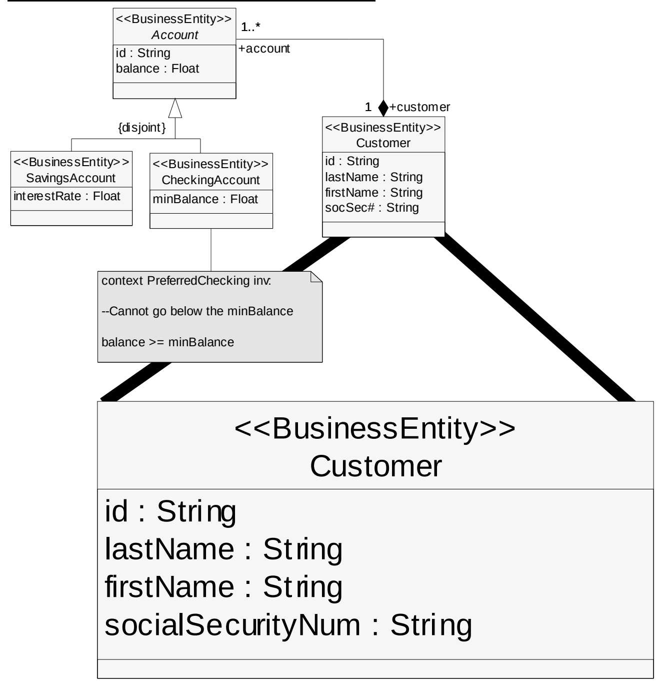
## Value Chain Contract

### How Clear?



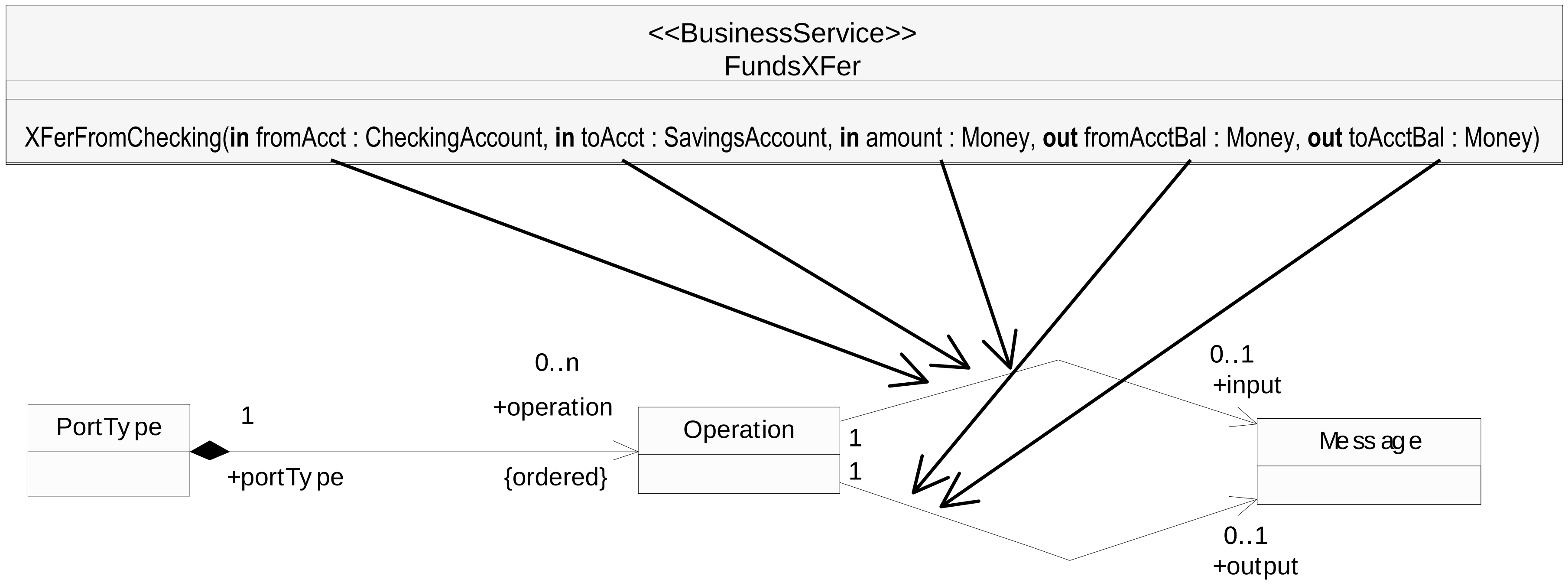
# Mapping the Business Information Model to XML





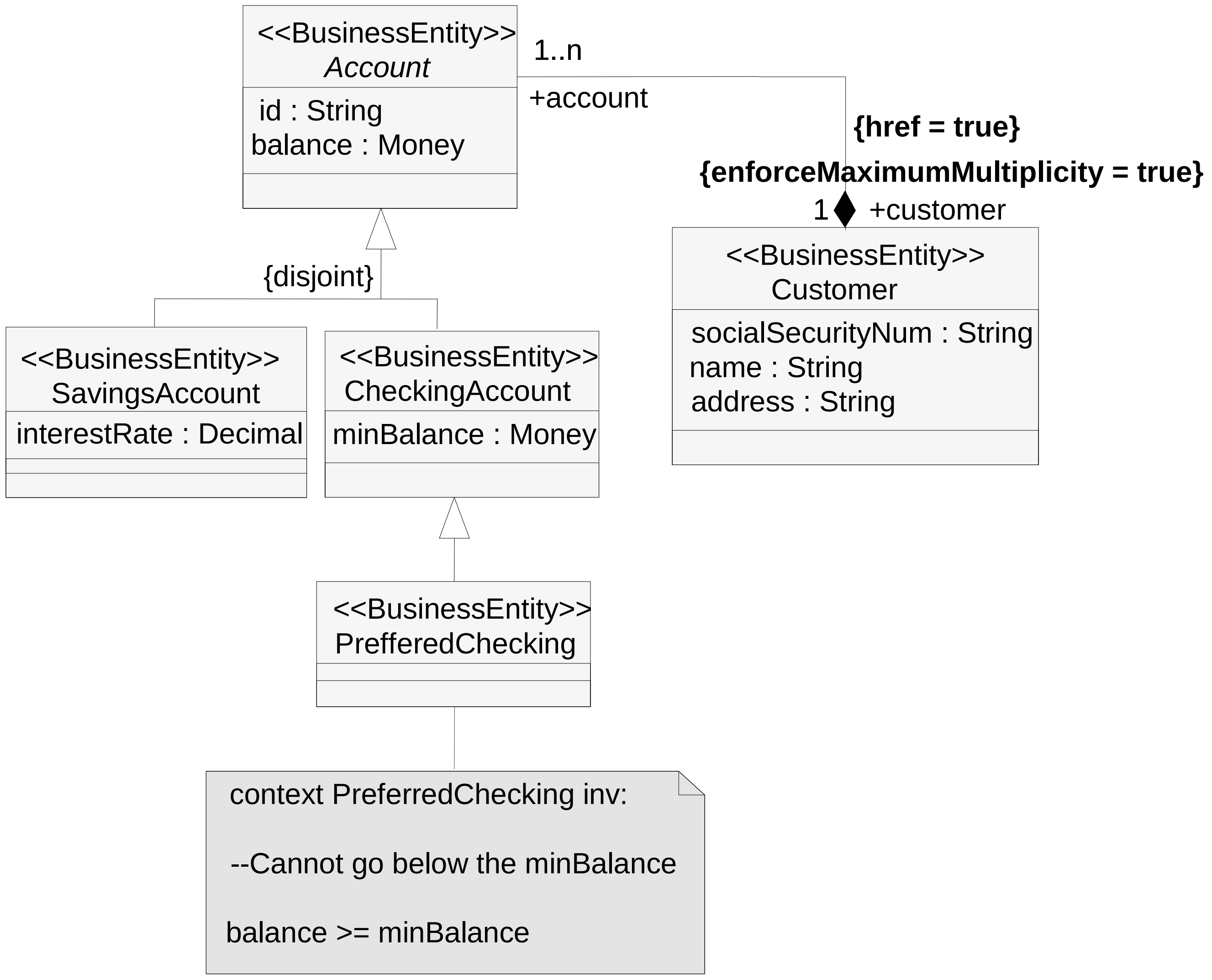
```
…
<!ELEMENT Bank.Customer.id (#PCDATA | XMI.reference)* >
<!ELEMENT Bank.Customer.lastName (#PCDATA | XMI.reference)* >
<!ELEMENT Bank.Customer.firstName (#PCDATA | XMI.reference)* >
<!ELEMENT Bank.Customer.socialSecurityNum (#PCDATA |
XMI.reference)*>
 …
```

# Mapping the Business Service Model to WSDL



<<BusinessService>>
FundsXFer
XFerFromChecking(**in** fromAcct : CheckingAccount, **in** toAcct : SavingsAccount, **in** amount : Money, **out** fromAcctBal : Money, **out** toAcctBal : Money)
PortType
1
+portType
0..n
+operation
{ordered}
Operation
1
1
0..1
+input
Message
0..1
+output

# Parameterized Mappings



context PreferredChecking inv:

--Cannot go below the minBalance

balance >= minBalance

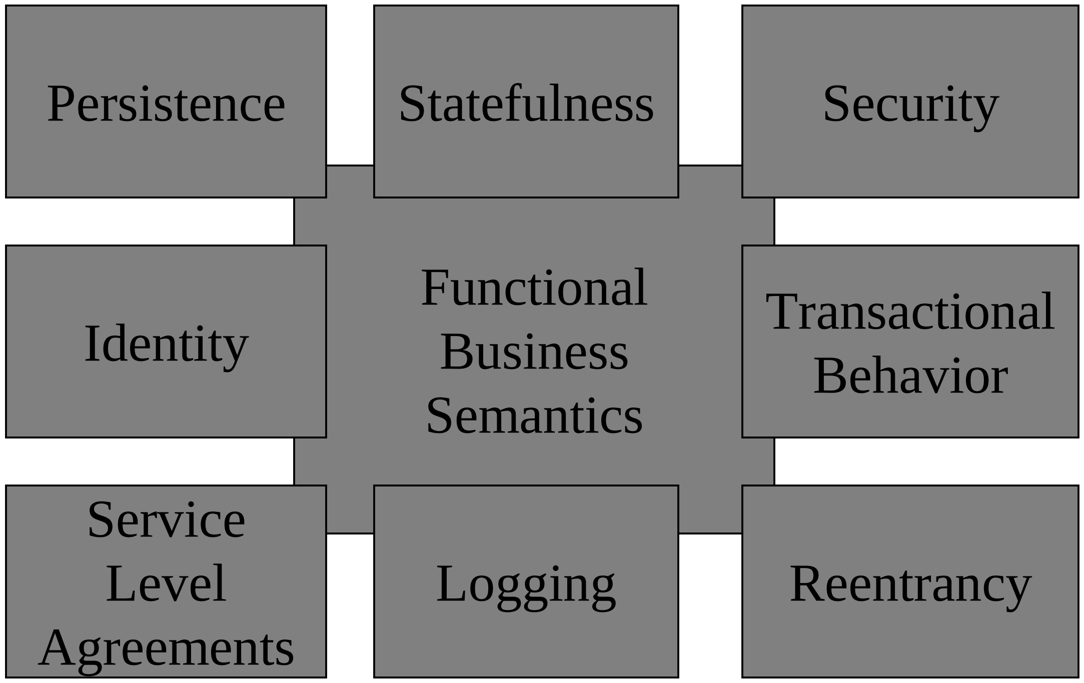
## Agenda

- The Demands of the Virtual Enterprise
- MDA: Industrializing Software
- Informal vs. Formal Modeling
- ➡ Future MDA Directions

## Related Technologies

- Aspect-Oriented Modeling
- Product Line Practices
- Intentional Programming
- Generative Programming
  - Key book: *Generative Programming*, Krzysztof Czarnecki and Ulrich W. Eisenecker
- Microsoft modeling directions
  - Key book*: Software Factories: Assembling Applications with Patterns*, *Models, Frameworks and Tools*, Jack Greenfield et al (H1’04)

# Aspect-Oriented Modeling



- Separating different aspects of a system at design time
  - Related to Multidimensional Separation of Concern
- An approach to separation of concern
- Addresses “code tangling” problem

## Product Line Practices

- *Product Line*
  - "…a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. "—Carnegie Mellon Software Engineering Institute
- *Core Asset Development*
  - Capture domain knowledge in the form of reusable assets
    - Define the scope of the domain
    - Model the domain
    - Develop components
    - Define an architecture
- *Production Plan*
  - How to produce systems using the core assets
- *Product Development*
  - Uses core assets according to the production plan
  - Creates individual products

## Intentional Programming

- Objective: “Make the source look like the design”
- Programming via *intentions*
  - High-level abstractions
- Active Source
  - Knows how to compile itself, support editing, rendering, and debugging
    - Behaviors called at programming time
- Source graph
  - Each abstract syntax tree (AST) node has a link to its metadata
  - The primary representation that plug-in modules deal with
    - Not text or graphics vectors
- Transformations from one level of abstraction to another

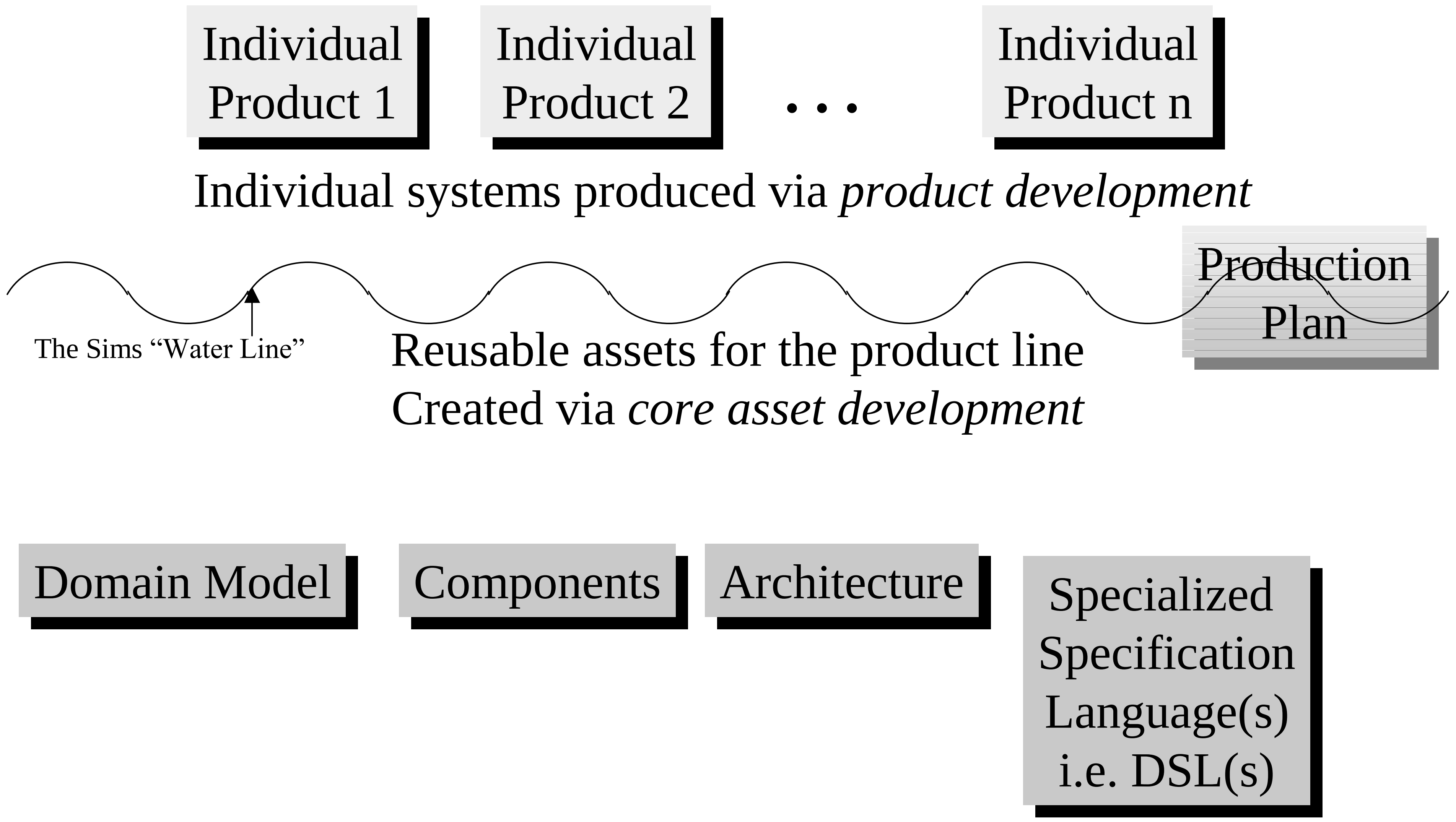
Charles Simonyi’s new company: *Intentional Software*

## Generative Programming (GP)

- Synthesis of
  - Aspect-Oriented Modeling
  - Product Line Practices
  - Intentional Programming
- Product Line Practices extended to include specifying Domain-Specific Languages (DSLs) as core assets for a product line
- Different DSLs for different aspects of the system
- Generators encapsulate product line knowledge
  - Transformations from one level of abstraction to another
- Extensible development environment based on common technology for representing source graphs in memory
  - Capable of hosting active source for multiple DSLs
- Model Integrated Computing
  - Vanderbilt Institute for Software Integrated Systems

# Product Line Practices

## Extended to Include Domain-Specific Languages


Individual Product 1
Individual Product 2
. . .
Individual Product n
Individual systems produced via *product development*
Production Plan
The Sims "Water Line"
Reusable assets for the product line
Created via *core asset development*
Domain Model
Components
Architecture
Specialized Specification Language(s) i.e. DSL(s)

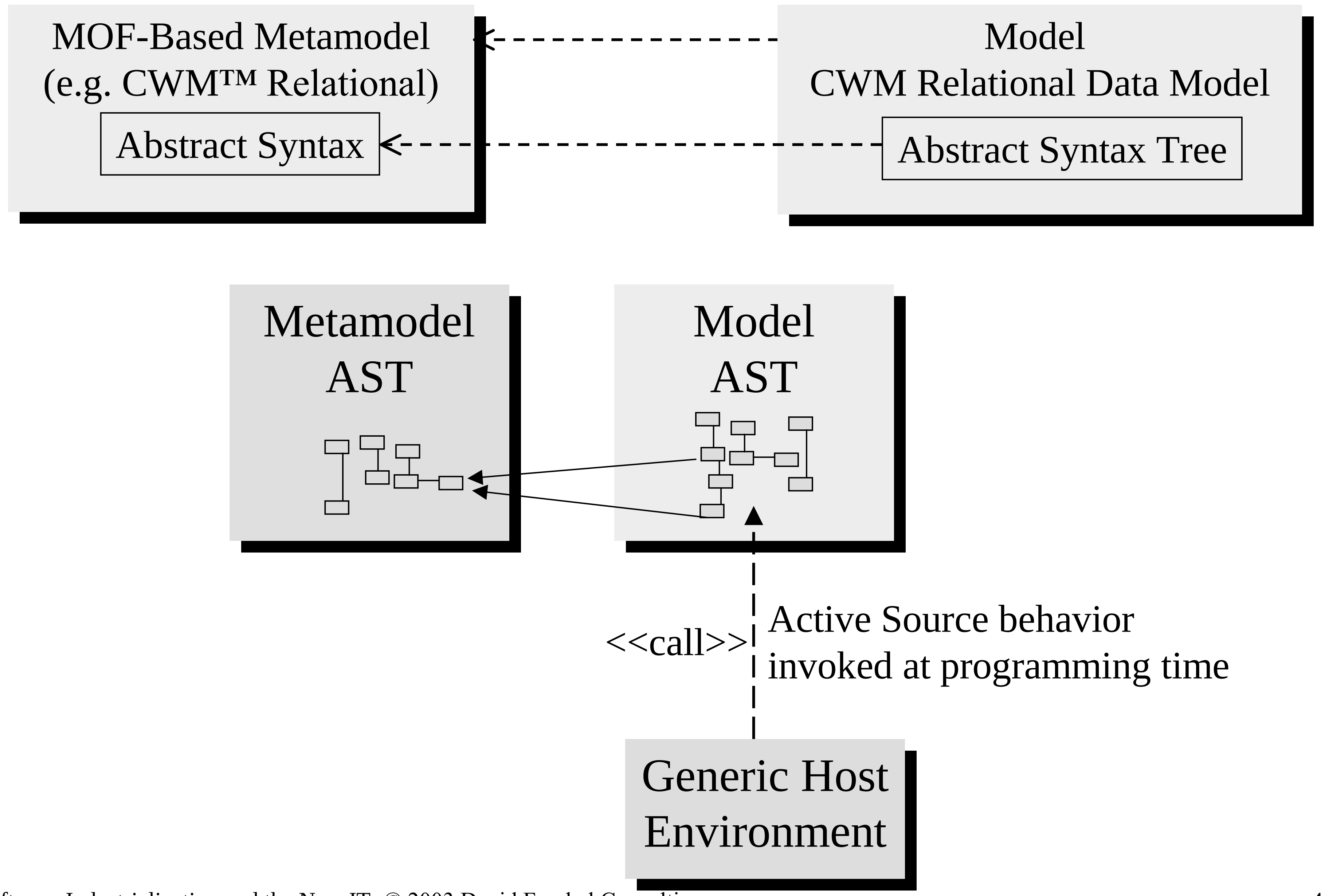
## Generative Programming (GP)
Design Time Composability

- Component description in some DSL pulled in at design-time
  - Application-specific configuration added
- Generator produces tailor-made component with minimal foot print
- Similar to the latest manufacturing processes
- "Just-In-Time CBD"

# MDA as a Standards Base for Product Lines and Domain-Specific Languages

- Domain Specific Languages
  - Languages defined via Meta Object Facility (MOF)
  - MOF-HUTN specification for textual DSLs
  - MOF lacks the ability to define graphical syntaxes
- Active Source
  - MOF-defined language packaged in a *modeling framework* with components, editor, generator, debugger, rendering support\
- Source graphs in extensible development environment for hosting active source
  - JMI, driven by MOF metamodels of each DSL
    - JMI provides link from an AST node to its metadata via MOF reflection

# Active Source Graphs



## MDA as a Standards Base for Product Lines and Domain-Specific Languages (continued)

- Definitions of generators
  - MOF Query View Transformations (QVT)
- Interchange of programs among tools when not “in-memory”
  - XMI, driven by MOF metamodels of each DSL

## Eclipse Modeling Framework as a GP Environment

- Already in place:
  - Ecore for defining abstract syntax
  - Java mapping for source graph (uses its own reflection, not JMI-MOF reflection).
  - XMI for interchange
- Still needed:
  - Ability to define textual DSLs on top of abstract syntax, using MOF-HUTN specification
  - Ability to define graphical DSLs on top of abstract syntax
    - Implement over GEF
    - DSTC project
    - Extensions to MOF standards to follow
  - Ability to define debugger plug-ins tied to abstract and concrete syntax

## MOF Industry Status

- New MOF-based initiatives
  - Business Process Definition Metamodel (OMG)
    - BPMI.org involved
  - Business Rules Metamodel (OMG)
    - Key people from business rules community involved
  - Ontology Definition Metamodel (OMG)
    - Key people from Semantic Web community involved
  - Distributed Management Task Force (DMTF)
    - Moving toward MOF-based metadata
  - Model-Driven data transformations a huge opportunity (CWM)—a killer app for MDA
- Microsoft committed to GP approach
  - But not to MOF
- MOF-Eclipse alignment is important

## Agenda Review

- The Demands of the Virtual Enterprise
- MDA: Industrializing Software
- Informal vs. Formal Modeling
- Future MDA Directions