MDA vision (*)

- To allow definition of *machine-readable application and data models* which allow long-term flexibility of:
  - *implementation*: new implementation infrastructure (the “hot new technology” effect) can be integrated or targeted by existing designs
  - *integration*: since not only the implementation but the design exists at time of integration, we can automate the production of data integration bridges and the connection to new integration infrastructures
  - *maintenance*: the availability of the design in a machine-readable form gives developers direct access to the specification of the system, making maintenance much simpler
  - *testing and simulation*: since the developed models can be used to generate code, they can equally be validated against requirements, tested against various infrastructures and can be used to directly simulate the behavior of the system being developed

(*) OMG MDA Guide v2.0 – June 2014

MDA overview

- The Model-Driven Architecture starts with the well-known and long-lasting idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform
- MDA provides an approach for, and enables tools to be provided for:
  - specifying a system independently of the platform that supports it
  - specifying platforms
  - choosing a particular platform for the system and
  - transforming the system specification into one for a particular platform
- The three primary goals of MDA are *portability*, *interoperability* and *reusability* through architectural separation of concerns

Solving the integration problem

- There will not be consensus on hardware platforms
- There will not be consensus on operating systems
- There will not be consensus on network protocols
- There will not be consensus on programming languages
- There will not be consensus on modeling languages

There must be consensus on the metamodel language!

Challenges with middleware

- A lot of standard middleware
  - SUN = Java RMI, EJB, JMS, J2EE, Jini …
  - Microsoft = OLE -> COM -> DCOM -> COM+ -> .NET
  - W3C = HTTP -> XML -> SOAP -> Web services
  - New ones appear, e.g. CCM, .NET, Web services, …
  - When old ones disappear, e.g. COM, DCE, …
- What is the “best” one?
- What is the “Next Best Thing”?
- How building software for the long term?
- How moving between component middleware?
- How preserving business application logic designs?
- How addressing the recurrent gap between business requirements and component middleware artifacts?

MDA solution

- Focus on Platform Independent Models (PIM)
  - without middleware details
- Abstract Platform Specific Models (PSM)
  - including all middleware details
- Define PIM to PSM transformations (i.e., set of rules and techniques to modify a PIM in order to get a PSM)
- Preserving PIM when new middleware appears!
- The shift is *from middleware to* ***modelware***

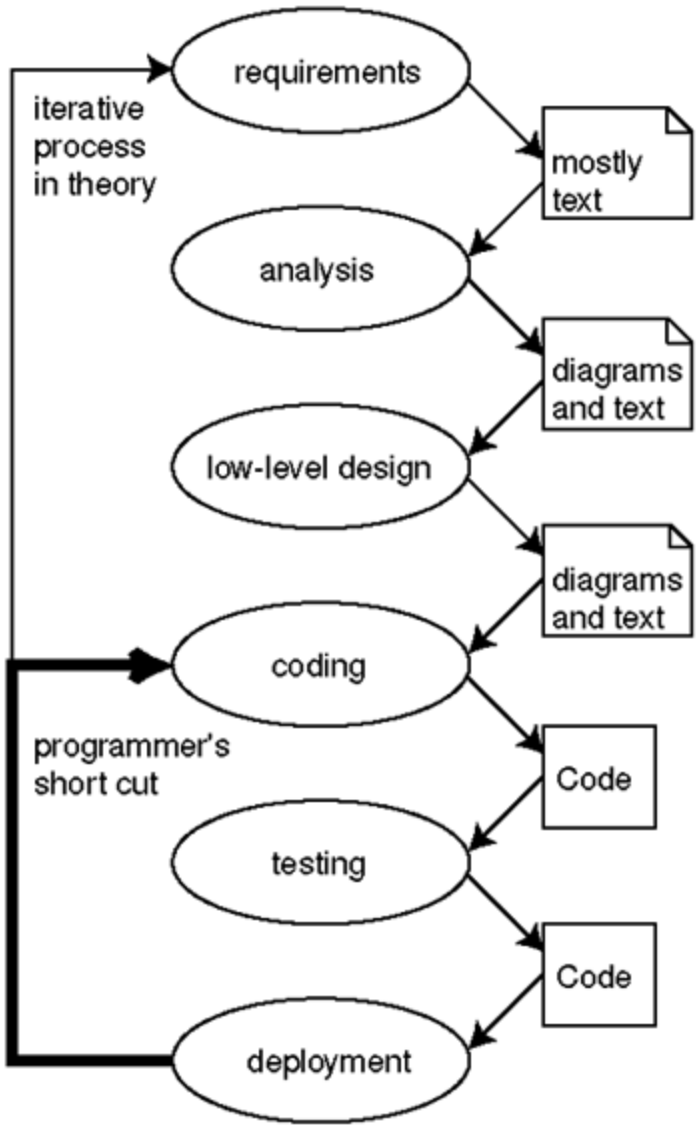
MDA viewpoints

- Computation Independent Viewpoint
  - The computation independent viewpoint focuses on the environment of the system, and the requirements for the system
  - The details of the structure and processing of the system are hidden or as yet undetermined
- Platform Independent Viewpoint
  - The platform independent viewpoint focuses on the operation of a system while hiding the details necessary for a particular platform
  - A platform independent view shows that part of the complete specification that does not change from one platform to another
  - A platform independent view may use a general purpose modeling language, or a language specific to the area in which the system will be used
- Platform Specific Viewpoint
  - The platform specific viewpoint combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system

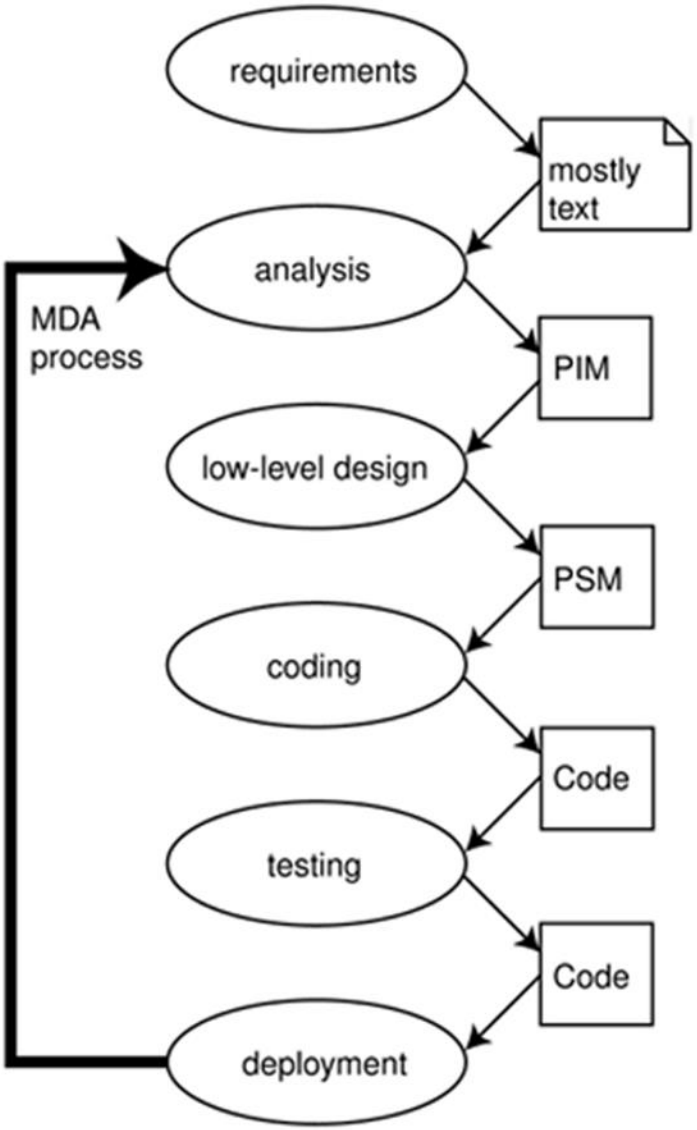
Views

- Computation Independent Model (**CIM**)
  - sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification
- Platform Independent Model (**PIM**)
  - a view of a system from the platform independent viewpoint
  - suitable for use with a number of different platforms of similar type
- Platform Specific Model (**PSM**)
  - combines the specifications in the PIM with the details that specify how that system uses a particular type of platform

Traditional sw development lifecycle



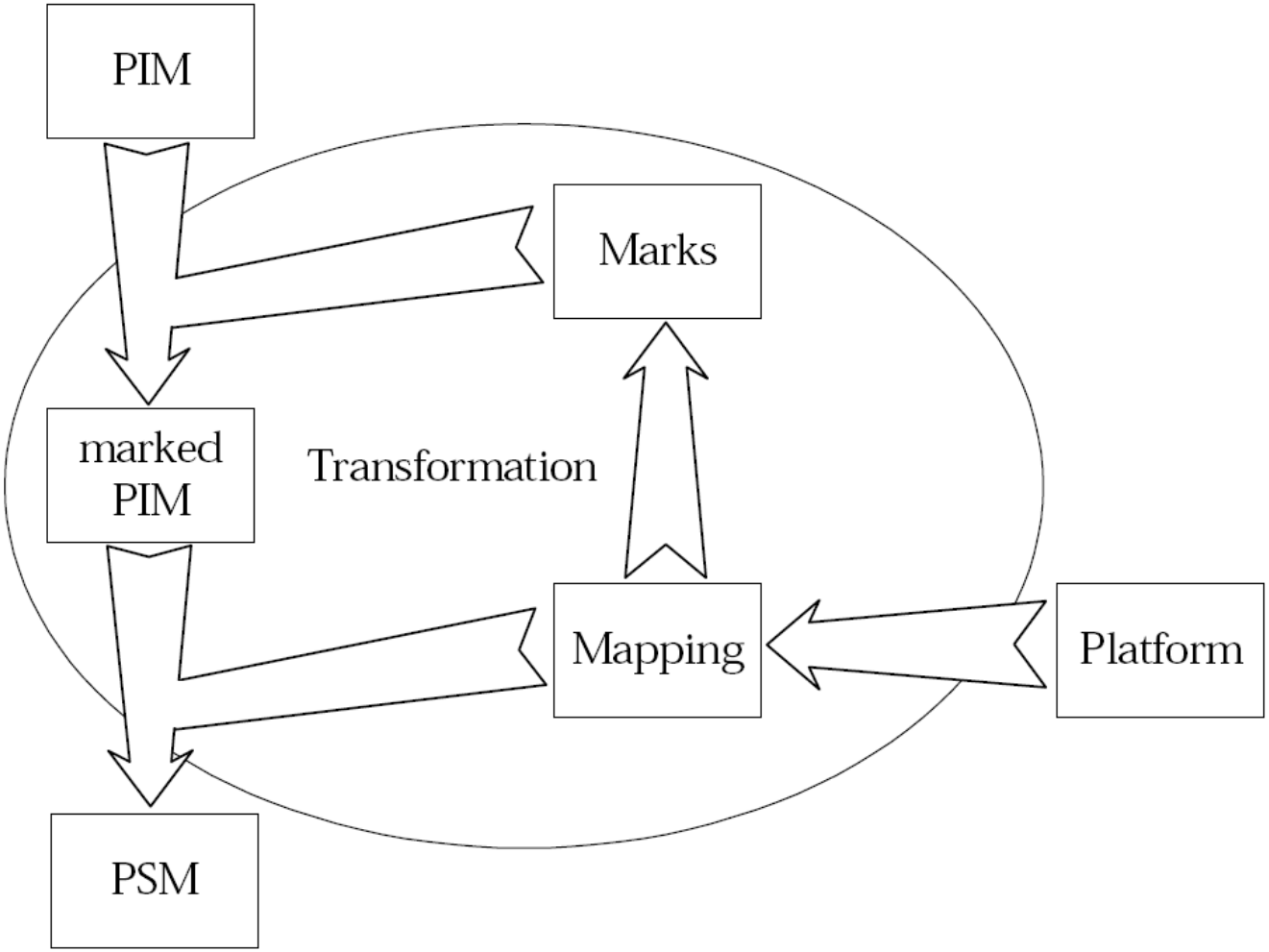
MDA development lifecycle



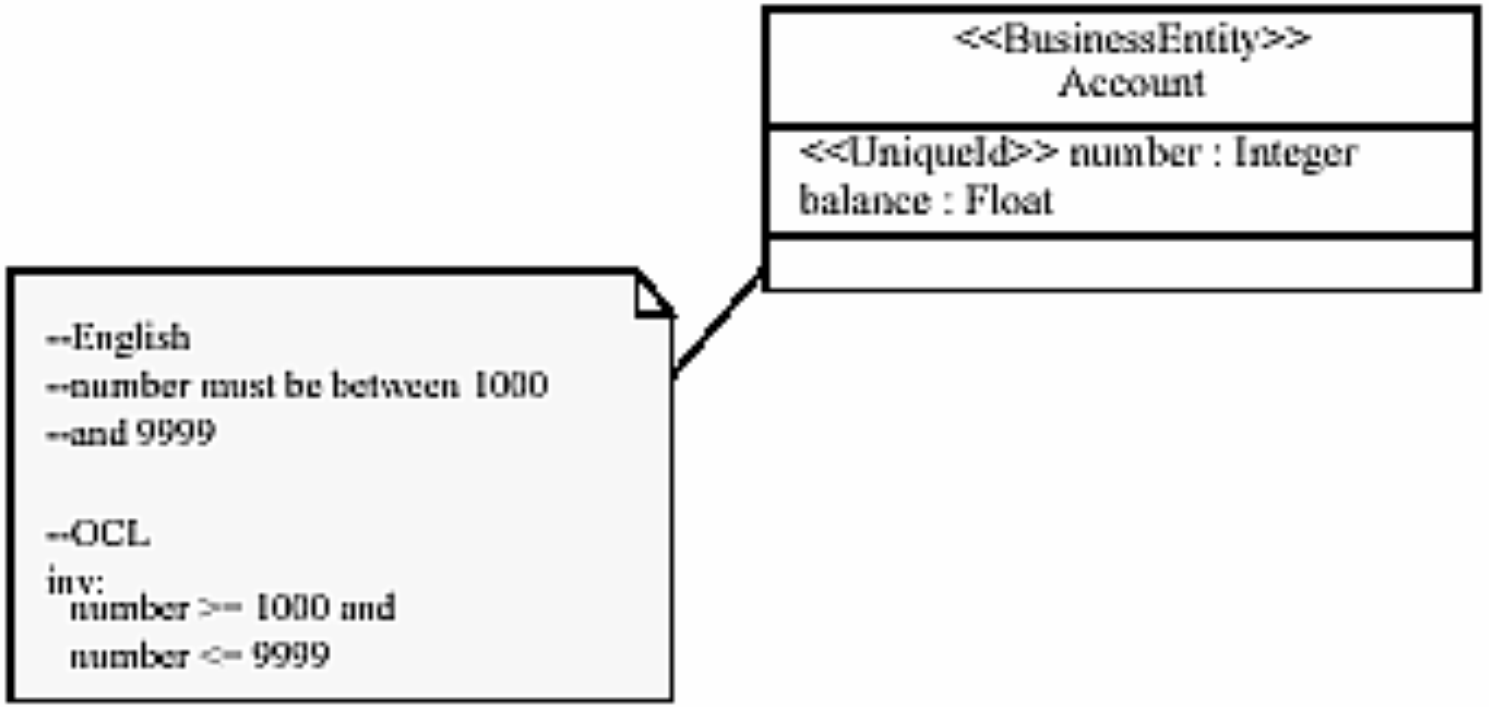
Mappings

- An MDA mapping provides *specifications for transformation* of a PIM into a PSM for a particular platform
- The platform model will determine the nature of the mapping
- A mark represents a concept in the PSM, and is applied to an element of the PIM, to indicate how that element is to be transformed
- *Example:*
  - *Entity is a mark that can be applied to classes or objects in a PIM*
  - *This mark indicates that an object marked as Entity in the PIM corresponds, in a PSM of CORBA Component type, to two objects, of types HomeInterface and EntityComponent, with certain connections between those objects.*

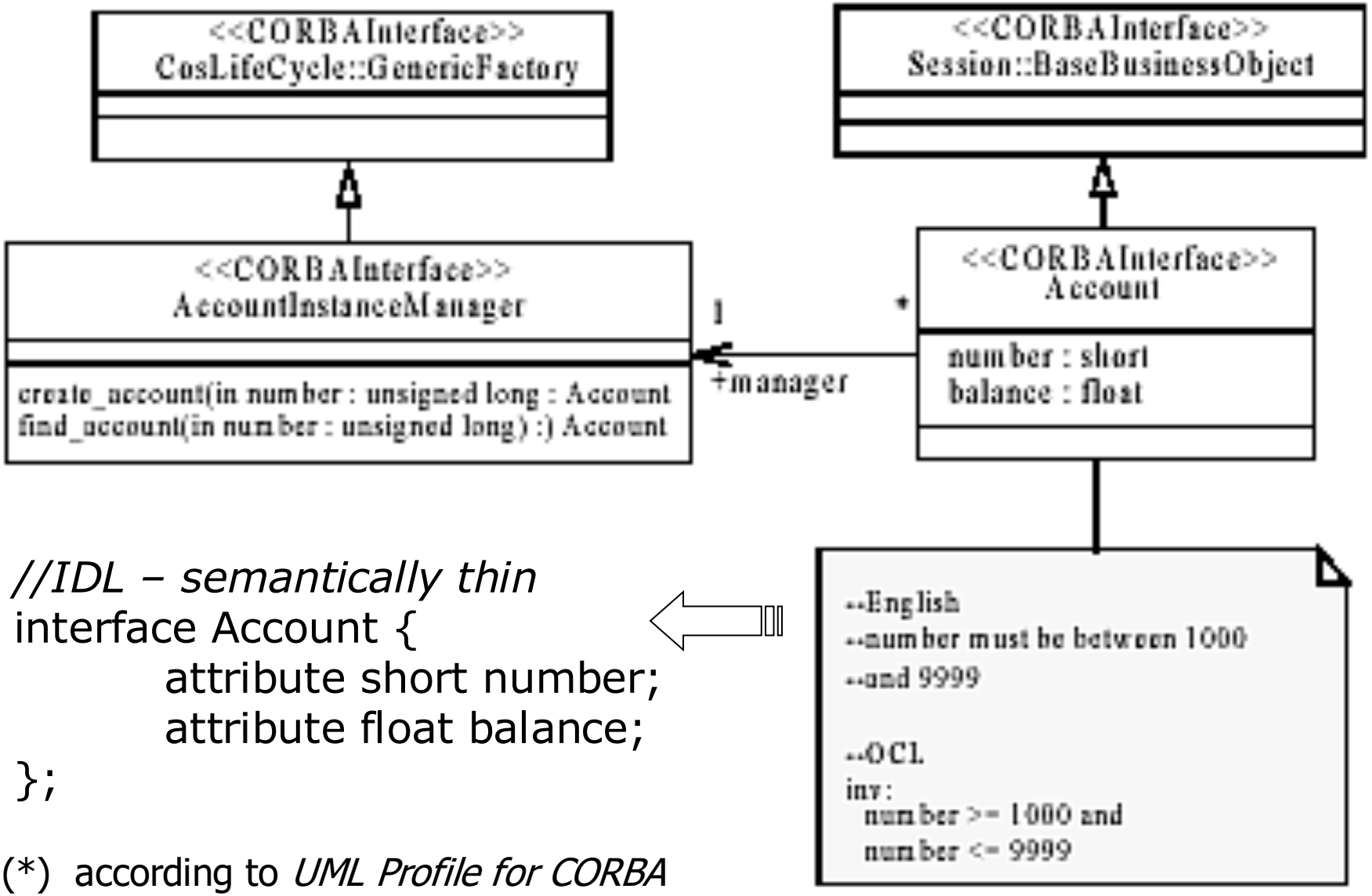
Model Transformation
(*marking approach*)



From a PIM...


<<BusinessEntity>>
Account
<<UniqueId>> number : Integer
balance : Float
--English
--number must be between 1000
--and 9999
--OCL
inv:
number >= 1000 and
number <= 9999

...to the corresponding PSM (*)




```
//IDL – semantically thin
interface Account {
        attribute short number;
        attribute float balance;
};
```

(*) according to *UML Profile for CORBA*

MDA in a nutshell