

Structuring Complexity of Business Applications



Björn Müller, SAP AG



© SAP AG 1998 K6 SAP TechEd '98, Karlsruhe (B. Müller) / 1

About this presentation...

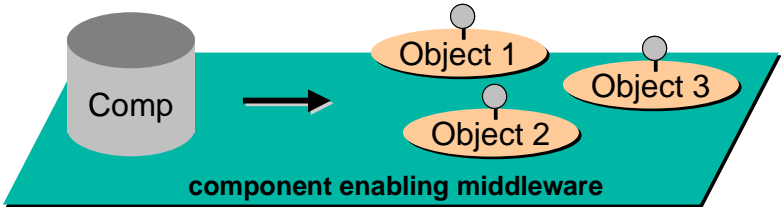
- **What this presentation will be about...**
 - Defining "Component"
 - Separation of concerns
 - Structuring a business application
 - Structuring business logic
- **... and what it will NOT be about!**
 - COM vs. CORBA
 - ABAP vs. Java vs. Visual Basic



© SAP AG 1998 K6 SAP TechEd '98, Karlsruhe (B. Müller) / 2

Component - A technical Definition

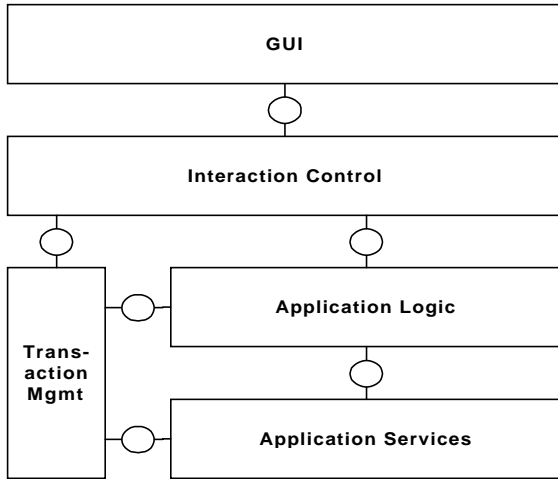
- A component is a piece of software which...
 - ... is **instanciated** at runtime one or more times. The instances are called "**objects**".
 - ... has a defined **interface**. The interface includes **properties**, **methods** and **events**.
 - ... which is **directly usable** (late binding).



Architectural Layers



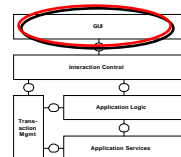
Architectural Layers (I)



Architectural Layers (II)

● GUI

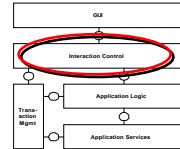
- Tasks
 - ◆ Rendering
Providing for GUI controls
Layouting an object with state
- Independent from...
 - ◆ Application Logic
Transaction Management



Architectural Layers (III)

- **Interaction Control**

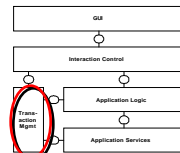
- Tasks
 - ◆ Defines “logical form” on business logic
 - Defines begin + end of transactions
 - Provides for answers on GUI specific questions
- Independent from...
 - ◆ GUI controls, GUI layouting
 - Semantic of application logic



Architectural Layers (IV)

- **Transaction Management**

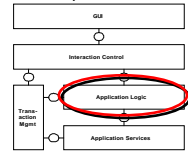
- Tasks
 - ◆ Providing for begin + end of transactions
 - Looking up objects to operate on
 - Embed base services into transaction management
- Independent from...
 - ◆ Application logic
 - Inner semantic of applications services



Architectural Layers (V)

- **Application Logic**

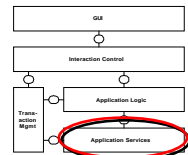
- Tasks
 - ◆ Business Rules, Business Functions
... across related Business Objects
- Independent from...
 - ◆ GUI
Transaction Management (with exceptions...)
Persistency/ Authority Mgmt/ ... (Application Services)



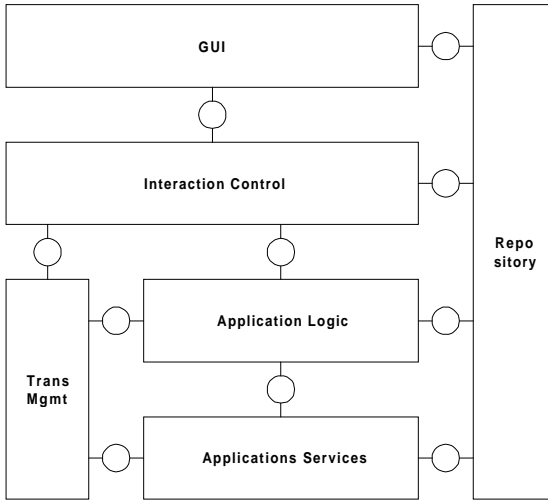
Architectural Layers (VI)

- **Application Services**

- Examples
 - ◆ Persistency
Authority Mgmt
Reporting
Tracing
 - ◆ Have descriptive/ structural knowledge of business logic.



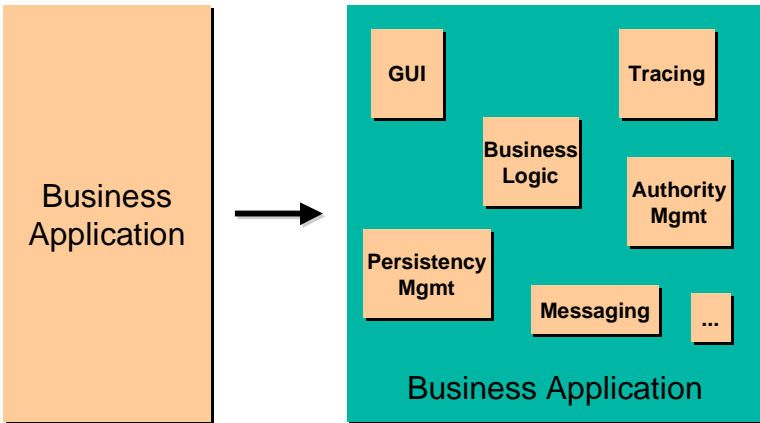
Repository



Decoupling Application Services from Application Logic

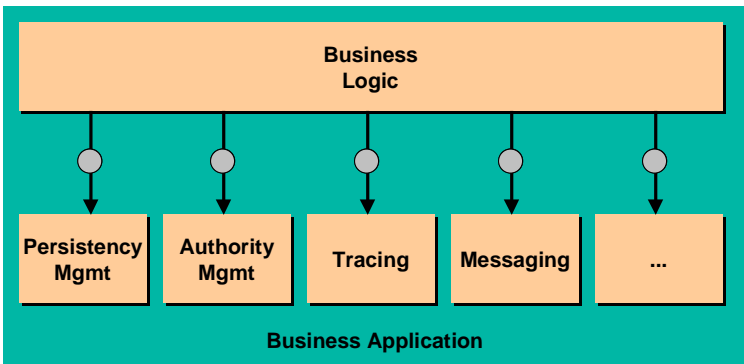


Guiding Principle - Separation of Concerns



Traditional way of separating concerns (I)

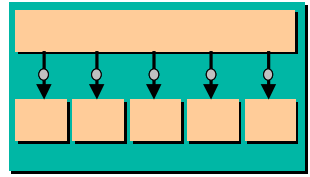
- The Business Logic has to be aware of its surrounding services and invokes them using defined interfaces.



Traditional way of separating concerns (II)

- **Positive**

- There **are** defined interfaces to common services!
- Services can be exchanged as long as they do not require interface changes!



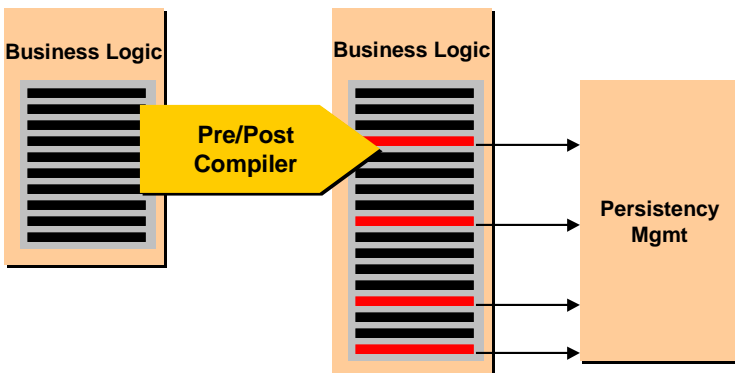
- **Negative**

- The business logic must be completely aware of its surrounding services.
 - ◆ New services mean changes in business logic.
 - ◆ Changes in interfaces have massive implications on business logic!



Approach "Generation" (I)

- **Example: binding to object oriented databases**



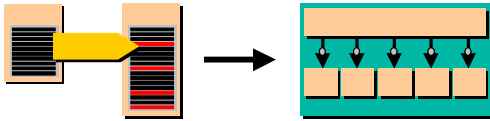
Approach “Generation” (II)

- **Positive**

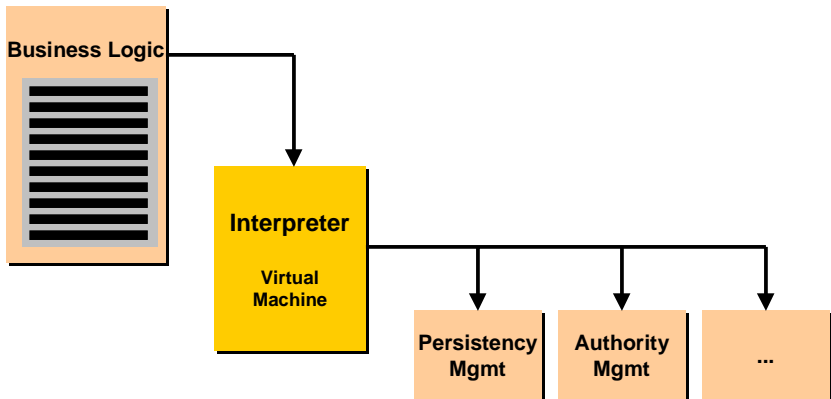
- Allows to implement business logic without being aware of services.

- **Negative**

- No proper definition of interfaces between business logic and service.
- Complex generation in cases of many services.
- Adding a new service/ Changing an existing service means that the whole business logic has to be recompiled.



Approach “Enhanced Interpreter” (I)



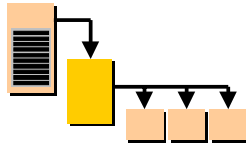
Approach “Enhanced Interpreter” (II)

- **Positive**

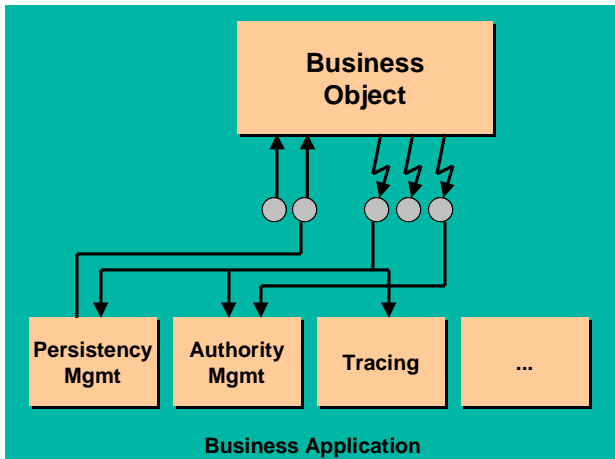
- True decoupling of business logic from services!
- New service/ Change of existing service will not have any impact on business logic.

- **Negative**

- Interpreter gets jobs which it is not designed for.
- Customers are bound to specific interpreter and can not use the interpreter they prefer.

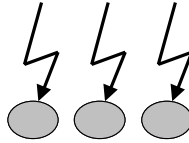


Approach “Eventing” (I)

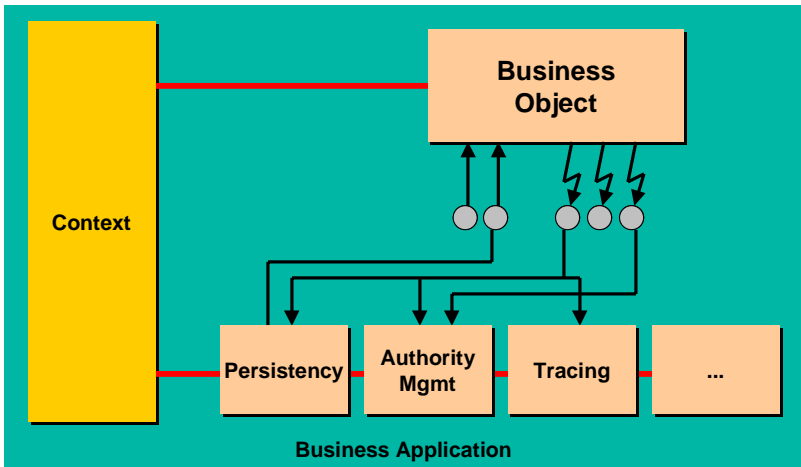


Approach “Eventing” (II)

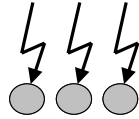
- “Events are method calls.”
- Events telling about access/ changes in state of object
 - StatePreAccessed
 - StatePreChanged, StatePostChanged
 - StatePreAdded/ StatePreRemoved (for collections)
- Events telling about interaction with object
 - MethodStarted
 - MethodEnded



Approach “Eventing” (III)



Approach “Eventing” (IV)



- **Positive**

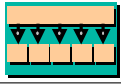
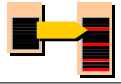
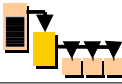
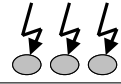
- True decoupling of business logic from services!
- New service/ Change of existing service will not have any impact on business logic.
- No dependency from any underlying infrastructure.

- **Negative**

- Do we really find the proper set of events to satisfy all business related services?



Different Approaches - Overview

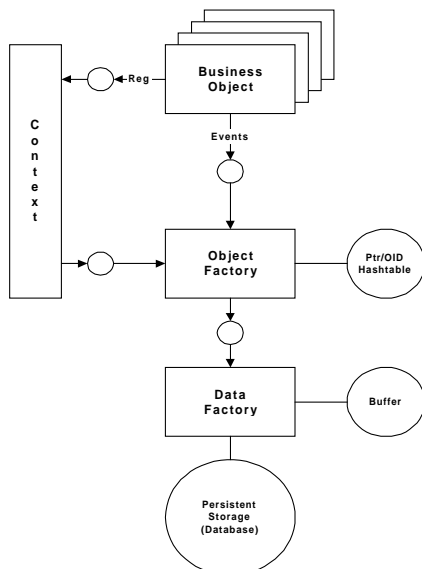
| |  |  |  |  |
|-------------|---|---|---|---|
| Decoupling | -- | - | ++ | ++ our choice |
| Our Ranking | 4 | 3 | 2 | 1 |



Application Service “Persistency Management”



Persistency Management (I)



Persistency Management (II)

● OBJECT FACTORY

- Responsible for bringing objects into memory
- Keeps control that no object is instanciated twice
- Marks manipulated objects as dirty
- Communicated to DataFactory to read and write data

● DATA FACTORY

- Responsible for storing data in a persistent way
- Has got buffer for performance enhancement and for sophisticated transaction management
- Concrete implementation: simple object relational mapping, 1 table represents 1 class



Persistency Management (III)

- **In principal any storage medium with transactional consistency can be used for persisting objects.**
- **We use relational databases. Reasons:**
 - Proven reliability
 - Ease of use and administration
 - Performance (dependent on usage!)



Object Relational Mapping (I)

- Approach “1:1 Mapping”

- 1 Class = 1 Table
- Fields in table correspond to class attributes
- Primary Key Field is Object ID which is unique in the local system. Object ID contains class name of object and system unique number (“com.sap.reservation.Reservation/217645638746534345”)
- Object links via foreign keys
- “Embedded relationships” = “Link relationships”



Object Relational Mapping (II)

- Approach “XML Stream”

- Object data is parsed into XML Stream:

```
<o>  
  <p>FirstName</p>  
  <v>Karl</v>  
  <p>LastName</p>  
  <v>Mustermann</v>  
</o>
```

- Embedded objects are part of XML Stream.
- XML Stream is stored in table as CLOB (Character LOB). Primary key field is Object ID.



Object Relational Mapping (III)

1:1 Mapping

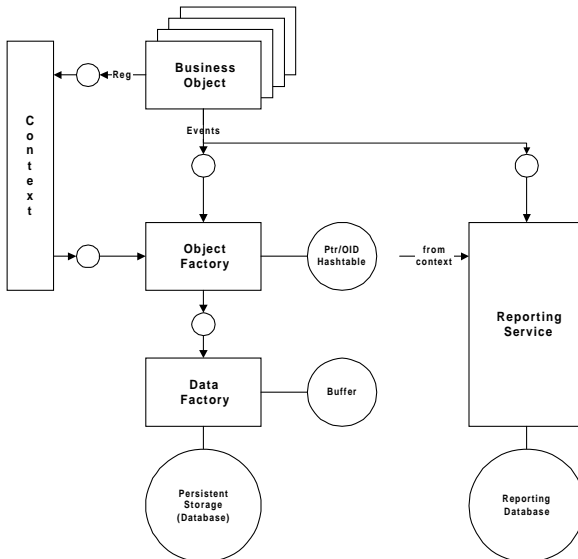
- **Advantages**
 - Fine granular updates of object data is possible.
 - Potentially possible: use of advances relational DB facilities
- **Disadvantages**
 - High number of DB operations for inserting complex objects.
- **Internal usage for...**
 - Customizing data, master data

XML Stream

- **Advantages**
 - Good performance
 - "Open" due to XML
- **Disadvantages**
 - Parsing of Stream not done by database
 - Database is used as "reliable filesystem"
- **Internal usage for...**
 - Business documents



Separation of Reporting Database (I)



Separation of Reporting Database (II)

● Object Orientation

- Single object as encapsulation of data and functions; encapsulation of responsibility
- Dividing up responsibility into sub-responsibilities: small granularity of single objects
- Navigation from one object to its related objects

● Reporting

- Mass of objects
- Aggregated view on data of various responsibilities
- Complex queries (SQL)



Separation of Reporting Database (III)

- **Reporting data is kept redundant from the beginning on.**
- **Reporting service is embedded into framework via eventing (“just another service”).**
- **Keeping reporting database separate allows us...**
 - ... to use a very simple OO relational mapping for the normal storage of objects
 - ... to change schemes of the normal storage of objects (e.g. to improve performance) without having to change reports



Performance

- **Keeping persistency as separate service allows us to centrally improve performance.**
 - Example: Persistency Management decides if to store objects as XML stream or if to store it via object relational mapping.
 - XML as “open protocol” for streaming of data allows to store business documents in a performant way.

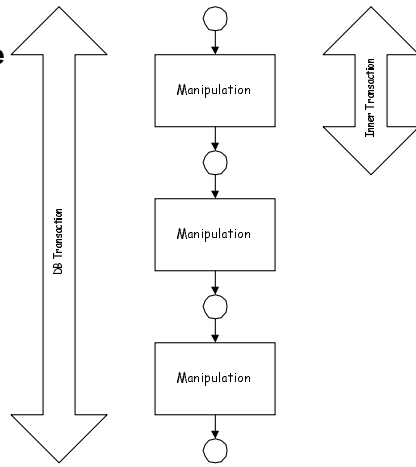


Transaction Management

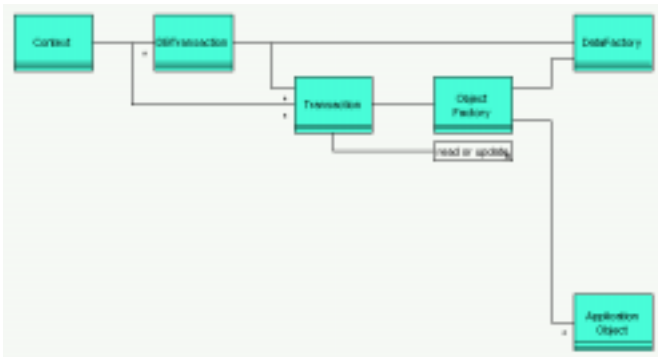


Motivation

- We want to deal with memory in the same way we deal with persistent storage - having a clear status management with commit and rollback behaviour.



Transaction Management (I)

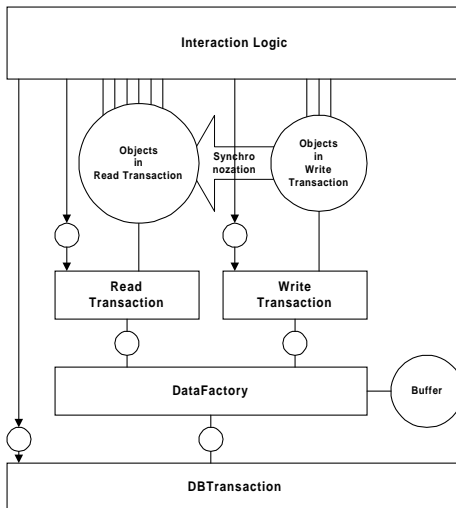


Transaction Management (II)

- **(Business) Objects are only valid within a transactional context.**
- **Manipulations to (Business) Objects are only possible within (Update) Transactions.**
 - Manipulations are performed on “explicit copies” (similar to check in/check out). Rollbacks just mean that the copies are released for garbage collection.
- **Objects in Read Transactions are synchronized if they are changed in Update Transactions.**
 - Types of synchronization:
 - ... by changes in own DBTransaction
 - ... by changes in other DBTransactions



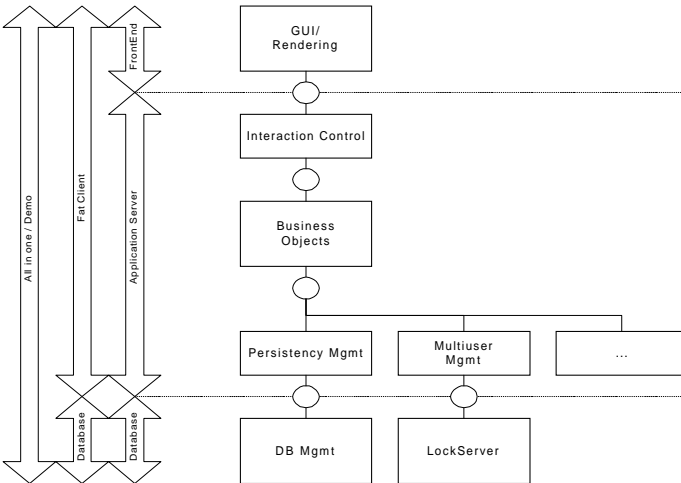
Transaction Management (III)



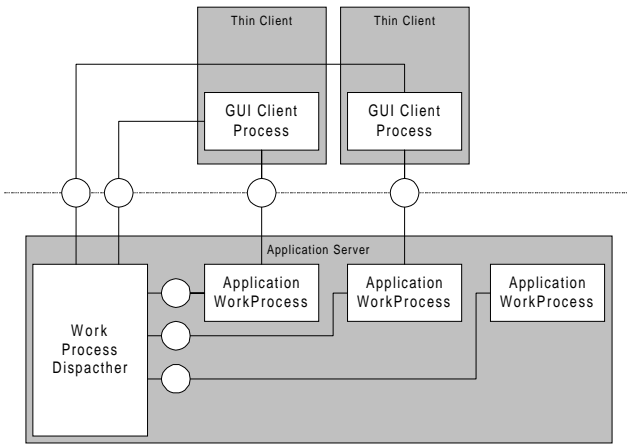
Client Server Architecture



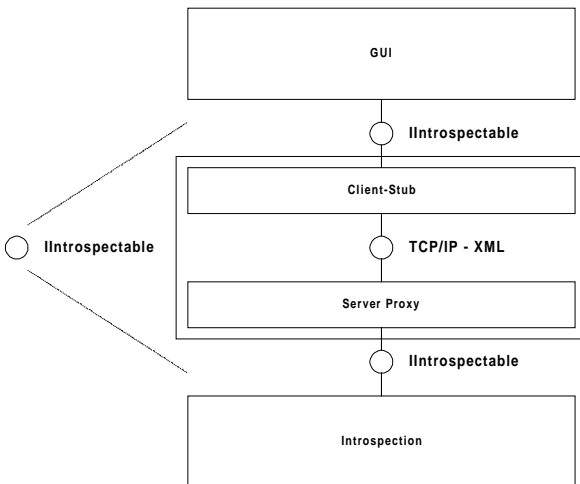
Client Server Architecture (I)



Client Server Architecture (II)

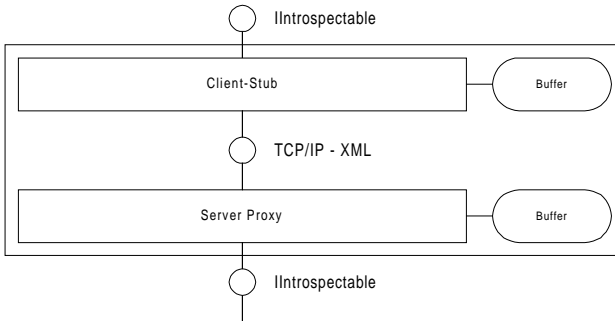


Decoupling GUI Layer (I)



Decoupling GUI Layer (II)

- **Buffering is inevitable in WAN environment. Network roundtrips must be reduced as much as possible.**



Decoupling GUI Layer (III)

- **“Decoupling the GUI Layer”...**
 - ... DOES NOT mean to split the GUI into a server and a client part.
 - ... DOES mean to manage object state in a client/server way.



Inside Business Logic...



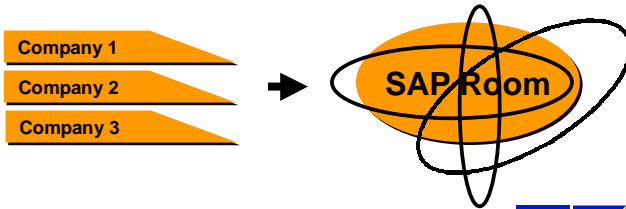
Typical Development Process

- Analyse your business processes (use cases).
- Find objects within processes.
Find out which data and behaviour objects have.
- Find objects with similar data and similar behaviour.
==> classes, inheritance
- Look on classes from an implementation point of view.
==> analysis => design



Problems (I)

- Different customers hold different opinions what a certain object really means to them
- Typical Reaction
 - The standard software provider adds all behaviour and data to the object which is “80%” of “all customer needs”
 - Consequence: Neither customer nor software provider are happy!



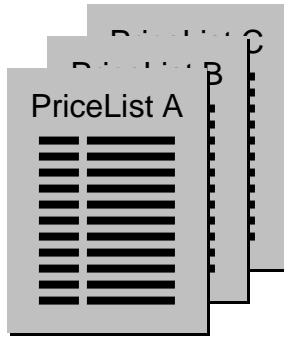
Problems (II)

- A standard software provider tends to define abstract classes in order to deliver a “ready to run” system.
 - Examples: material, resource
 - Consequence: customers are not happy with naming, they want to see “their objects” in their system



Problems (III)

- **Business behaviour often cannot be assigned to one object.**
 - Example: Whom do I ask for a price for a specific product?
 - Business behaviour is assigned to various relations of objects.

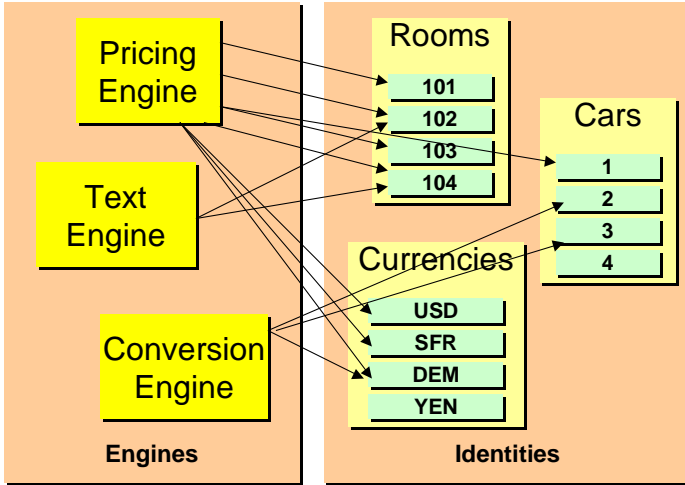


Conclusion

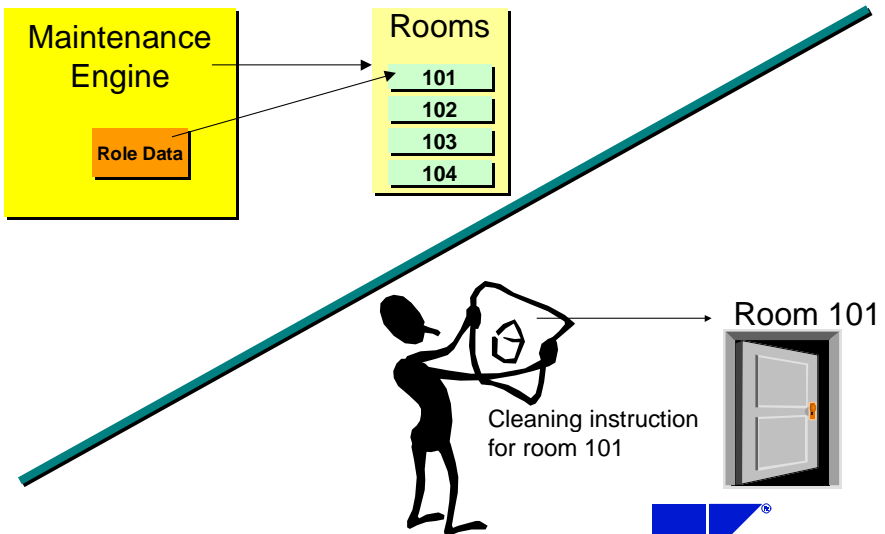
- **A standard software provider is NOT good in defining entities and their data + behaviour**
- **A standard software provider IS good in defining generic components representing a certain business responsibility**
- **Example: Pricing**
 - “We can build a pricing engine which allows to calculate prices for any objects based on rules customized within this pricing engine.”
 - The pricing engine does explicitly not know the business semantic of the objects which influence its pricing.



Separating Business Logic from Identities (I)

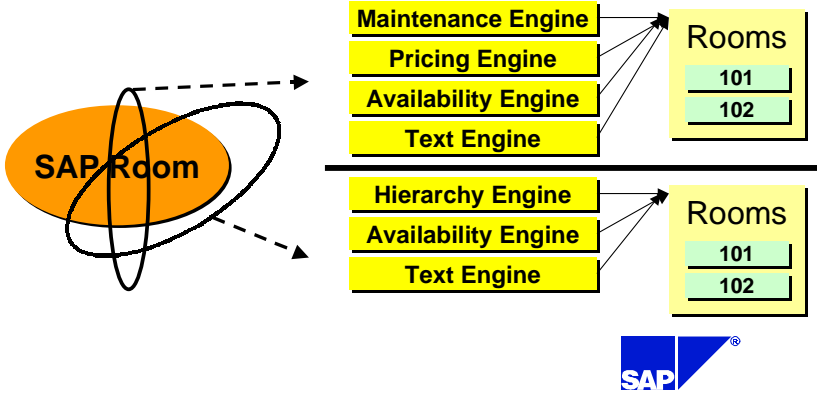


Separating Business Logic from Identities (II)

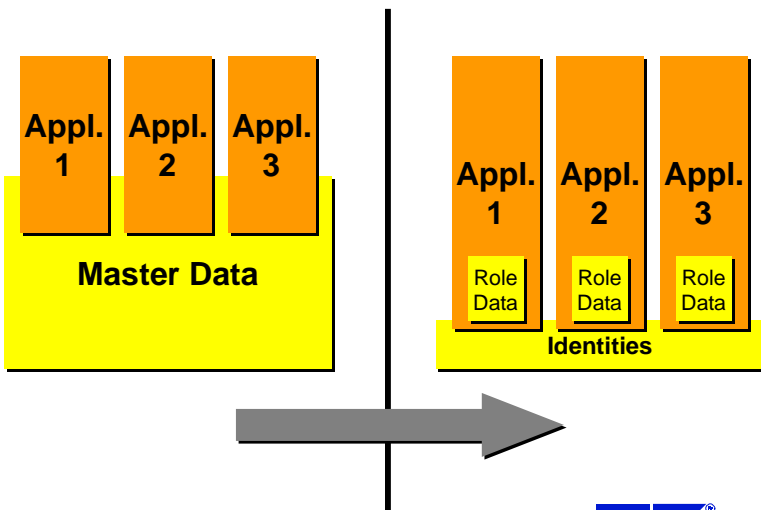


Separating Business Logic from Identities (III)

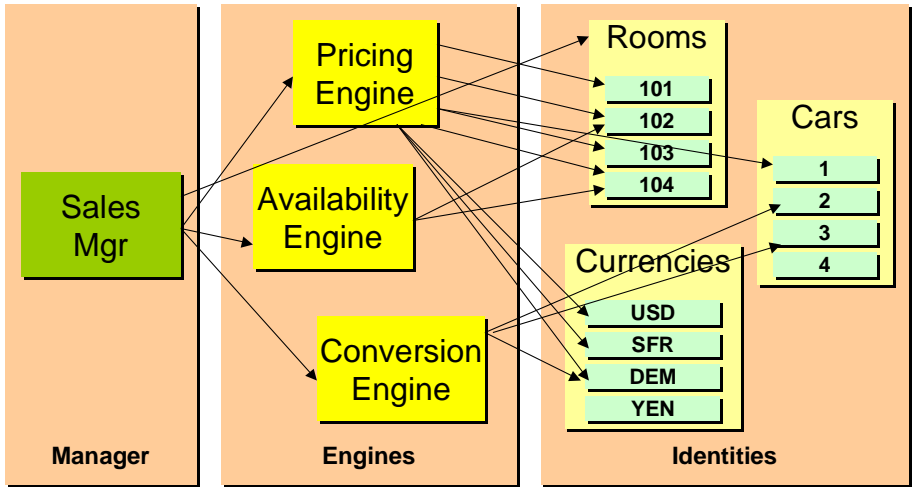
- Customers define the roles a certain identity plays in different engines.



Different View on Master Data



Manager Objects (I)

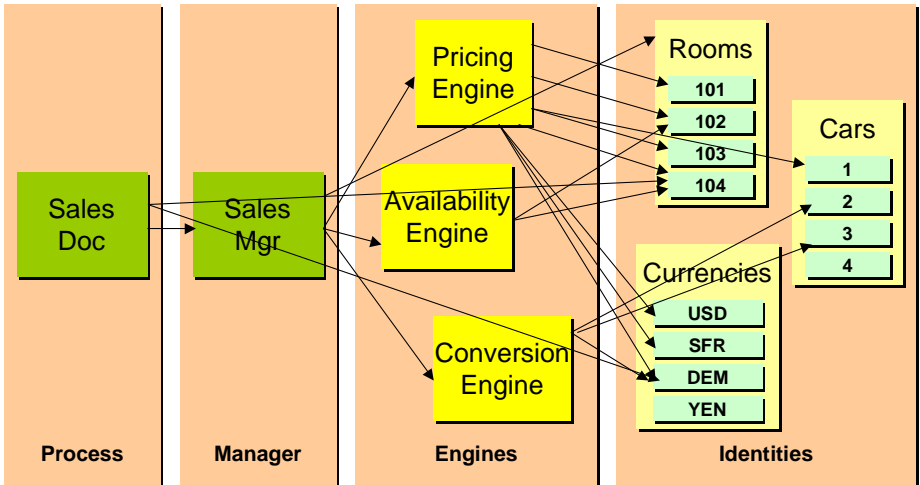


Manager Objects (II)

- **Managers combine Groups & Engines in order to combine all business responsibilities which are used for a specific business process.**
- **Managers only hold “delegation logic” and do not hold any kind of business logic.**



Process Objects (I)



Process Objects (II)

- **Process Objects are instanciated per business transaction.**
- **Process Objects hold all the data belonging to the business transaction + control how and when this data is processed.**
- **Process Objects relate to a manager.**



Business Logic - Overall Picture

