

**38th Hawaii International Conference of System Sciences**  
Hawaii, January 3rd-6th, 2005

**Adaptable and Evolvable Software Systems**

## **The EntityContainer – an Object- Oriented and Model-Driven Persistency Cache**

Gernot Schmoelzer, Stefan Mitterdorfer, Christian Kreiner,  
Joerg Faschingbauer, Zsolt Kovacs, Egon Teiniker, Reinhold Weiss



**Institute for Technical Informatics**  
**Graz University of Technology**  
**AUSTRIA**

### **Overview**

---

- **Motivation**
- **Related Work**
- **Architecture**
- **Data modeling**
- **Design of the Entity Container**
- **Dynamic interface - example**
- **Type-safe access layer - example**
- **Backingstore and persistent storage**
- **Recursive EC composition**
- **Transaction concept**
- **Conclusion and Future Work**



Institute for Technical Informatics  
Graz University of Technology

<http://www.iti.tu-graz.ac.at>

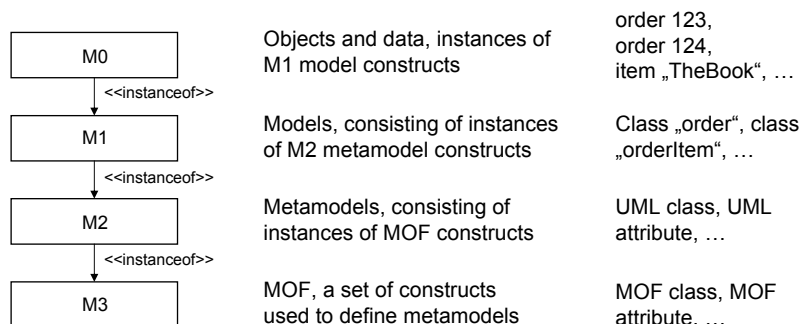
# Motivation

- **Requirements for Modern Software Development**
  - Abstraction to lower complexity
  - Paradigms, such as Object-oriented or Component-based SD
  - Persistency mechanisms (RDBMS, etc.)
  - Flexibility and extension mechanisms
  - High quality of software
- **Solution**
  - SD Process (Unified Process)
  - Model-Driven Development
  - Reuse of software (components, controls)
  - Testing (Test-driven Development)



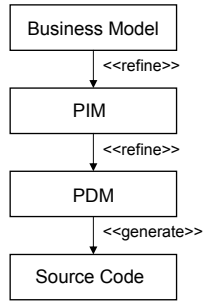
# Related Work – MOF Metadata Architecture

Four Layer Architecture [by OMG]

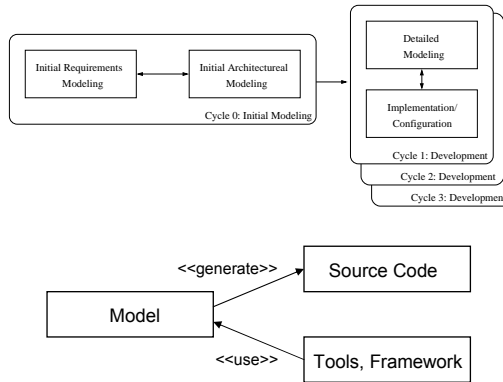


## Related Work – Model Driven Development

### MDA – Model Driven Architecture



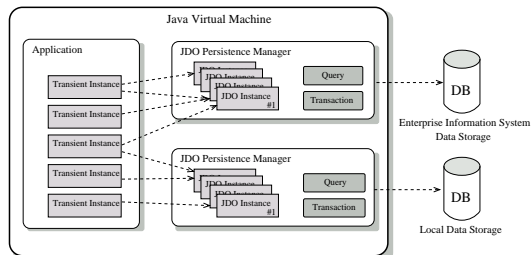
### AMDD – Agile Model Driven Development



## Related Work – Available Technologies (1/2)

### JDO – Java Data Object

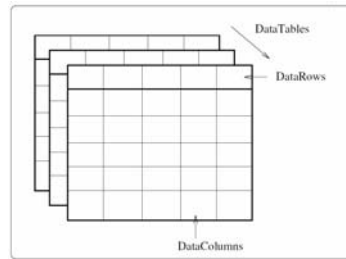
- **Advantages:**
  - OO data access
  - Transparent data store
- **Disadvantages:**
  - Mapping between objects in JVM and objects in data store manually
  - JDO object definition
  - Restriction to JAVA



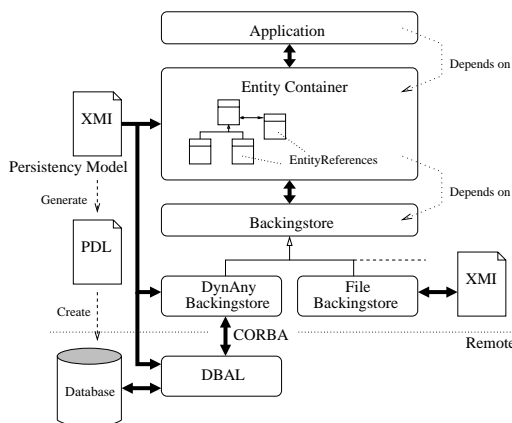
## Related Work – Available Technologies (2/2)

### ADO.NET – ActiveX Data Object

- **Advantages:**
  - In-memory DB
  - Transparent data store
- **Disadvantages:**
  - Relational programming model
  - No object-relational mapping
  - Restriction on platform (OS)



## Architecture

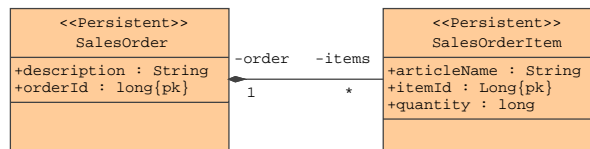


- Application only interacts with EC
- EC is based on a data persistency model
- Data model defines structure and constraints of persistent data
- Data model is designed at a higher abstraction level, independent of any persistency mechanism and programming language
- Data is checked against model according to structure and constraints
- Backingstore provides interface to different data storage systems

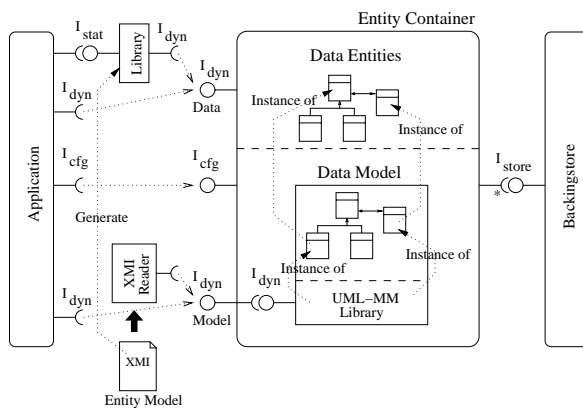


# Data modeling

- **Abstract system view**
- **Platform independent model**
- **Object-oriented paradigms (inheritance, etc.)**
- **Constraints definition**
- **Automatic mapping to persistency mechanism**
- **Automatic DB generation**
- **Common data interface**



## Design of the EC

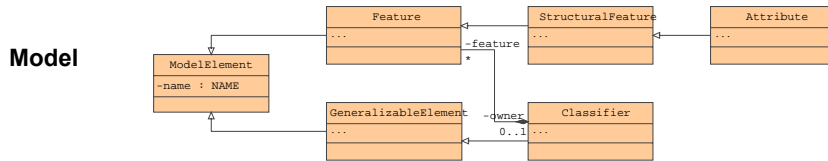


### Principles:

- **Each data entity object is based on (instance of) a data entity class**
- **Same dynamic interface is used to create data model as well as data entities**



## Creating data model



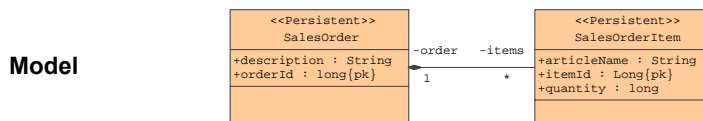
**Example**

```

// create model
IDynamicInterface ecMif = ec.getModelInterface();
ITemplate attr = new Template();
attr.setAttribute("name", new StringValue("orderId"));
ITemplate cls = new Template();
cls.setAttribute("name", new StringValue("SalesOrder"));
cls.setAssociation("feature", attr);
IEntityReference clsRef = ecMif.createByTemplate("Classifier", cls);
    
```



## Creating data entities



**Example**

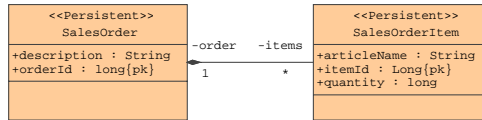
```

IDynamicInterface ecIf = ec.getDataInterface();
...
// create an order
ITemplate orderTempl = new Template();
orderTempl.setAttribute("orderId", new IntValue(123));
orderTempl.setAttribute("description", new StringValue("MyOrder"));
ITemplate bookTempl = new Template();
bookTempl.setAttribute("itemId", new IntValue(9823));
bookTempl.setAttribute("articleName", new StringValue("TheBook"));
bookTempl.setAssociation("order", orderTempl);
IEntityReference order = ecIf.createByTemplate("SalesOrder",
    orderTempl);
    
```



# Type-safe Access Layer

## Model



## Example

```
// create an order
SalesOrderClass f1 = ec.getSalesOrderClass();
SalesOrder order = f1.create(123);
order.setDescription("MyOrder");
// create an item for the order
SalesOrderItemClass f2 = ec.getSalesOrderItemClass();
SalesOrderItem item = f2.create(9823, order);
item.setArticleName("TheBook");
item.setQuantity(2);
```



# Data Retrieval

## Dynamic

```
// retrieve entity by using filter
IFilter filter = new AttributeFilter(
    "orderId", new IntValue(123),
    AttributeFilter.EQUAL);
IResultSet orders = ecIf.findByName("SalesOrder", filter);
...
// retrieve entity by using template
ITemplate templ = new Template();
orderTempl.setAttribute("orderId", new IntValue(123));
IEntityReference order2 = ecIf.findByPrimaryKey("SalesOrder", templ);
IResultSet items = order2.getConnections("items");
```

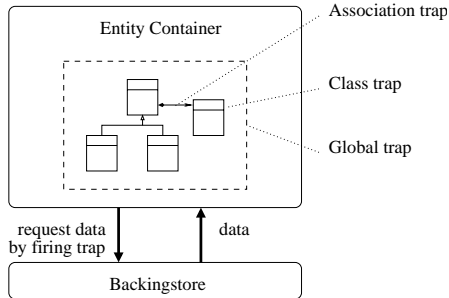
## Static

```
// retrieve entity by using filter
SalesOrderClass f1 = ec.getSalesOrderClass();
SalesOrder order = f1.findByPrimaryKey(123);
SalesOrderItem[] items = order.getItems();
```



# Backingstore and Persistent Storage

## Trap mechanism



## Entity states:

- NEW
- DIRTY
- SYNC
- REMOVED

## Trap:

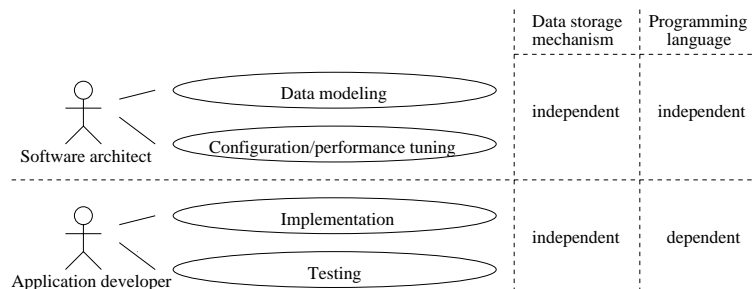
- specifies data fetched by the EC on a particular action
- Actions: global, class, association
- ONQN notation

Query example: `SalesOrder(orderId,description)[orderId=123].items[ ]`



# Development Process

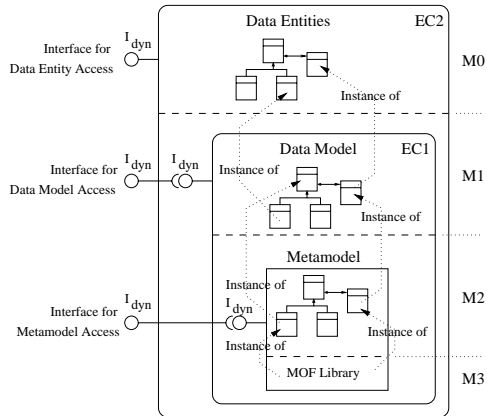
- Data management totally decoupled from specific persistency mechanism
- Allows to define different roles, depending on specialization
- Makes development more efficient and flexible
- Test concept based on EC



# Recursive Composition of ECs

## Equivalent Interfaces for all Model-layers

- EC covers three modeling layers
- Possible to cover the complete MOF architecture with two ECs



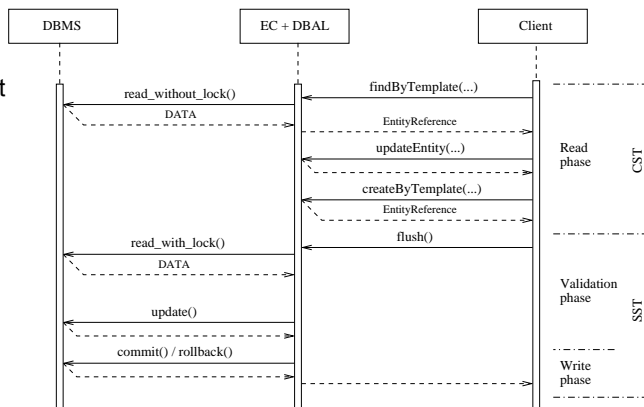
# Transaction Concepts / Optimistic Locking

## Sequence Diagram

Three phase concept

- Read phase
- Validation phase
- Write phase

• Also allows offline data access



## Adaptive programming based on EC

---

- **Static (type-safe) interface**
- **Dynamic interface for all modeling layers**
- **Allows to develop adaptive and flexible tools and frameworks based on a data model**
- **Allows meta-programming (similar to Java Reflection), but also read/write**
  
- **Possible Frameworks**
  - **for GUI controls**
    - visualization of data (e.g. table, tree)
    - checking of constraints
    - automatic navigation in the model
  - **for remote synchronization**
  - **for data access with weak connection**



## Conclusion and Future Work

---

- **Conclusion**
  - **Persistent Data Modeling**
  - **Object-oriented data access**
  - **Type-safe access layer**
  - **Validation against data model**
  - **Shared Transactions / optimistic locking**
  - **Different roles in development process**
  - **Adaptive programming and meta-programming**
  
- **Future Work**
  - **Transient model extensions**
  - **„Beaming“ of ECs (Synchronization according to Data Transfer Object Pattern)**
  - **Dynamic and automatic GUI on top of the EC**



# Discussion

---

**Thank You!**



Institute for Technical Informatics  
Graz University of Technology

<http://www.iti.tu-graz.ac.at>