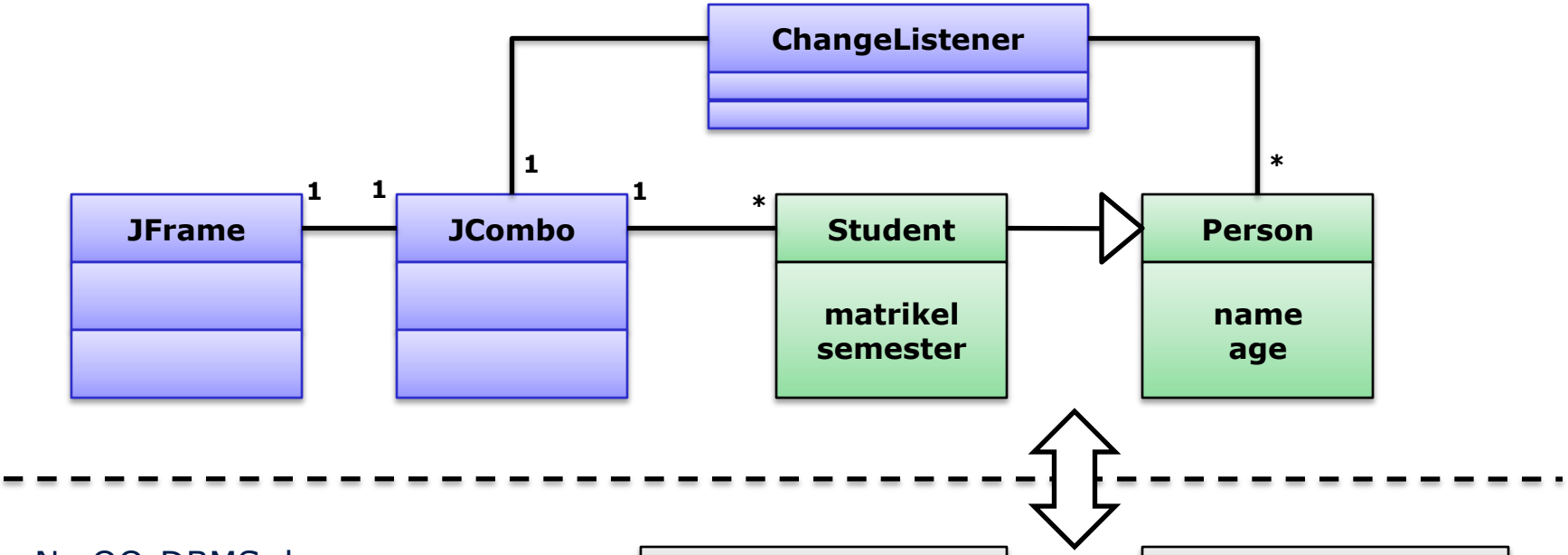
TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Software- und Multimediatechnik, Professur für Softwaretechnologie

Dresden Auto-Managed Persistence Framework

Defense of Diploma Thesis
Dresden, 18.03.2010, Sebastian Götz

Domain Object Persistency – Object-Relational Mapping

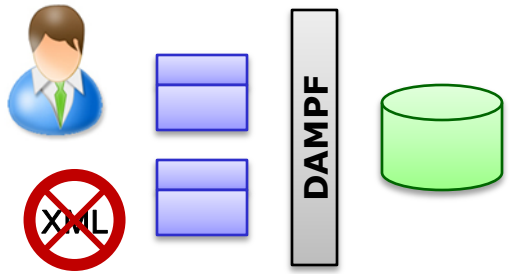


No OO-DBMS, because:

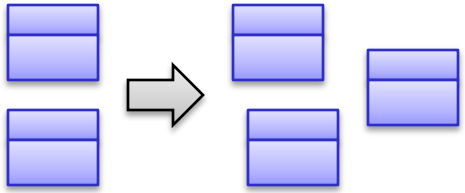
- Majority uses RDBMS
- No standards
- Not as performant

The Features of DAMPF:

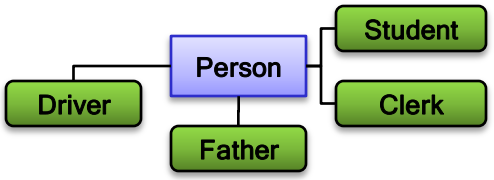
1. Transparency



2. Support for Schema Evolution



3. Support for Roles



Roles are a very new concept - since a long time.

Background: Introduction to Roles

John, a student, starts to work as a student assistant.

How to model such a scenario using Object-Oriented mechanisms ?

Single Inheritance

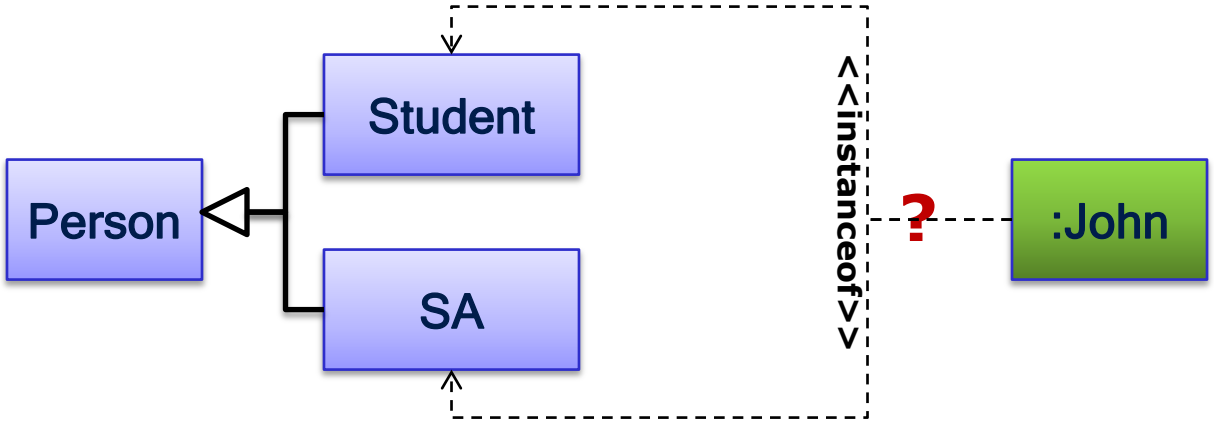
Multiple Inheritance

Delegation

Background: Introduction to Roles

John, a student, starts to work as a student assistant.

1. Approach: Single Inheritance

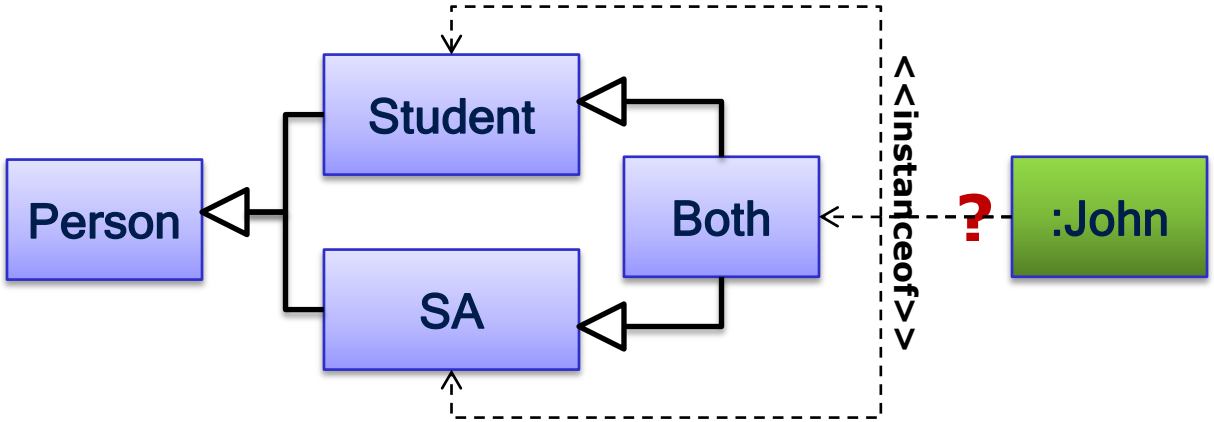


Instance cannot be of two types!

Background: Introduction to Roles

John, a student, starts to work as a student assistant.

2. Approach: Mutliple Inheritance

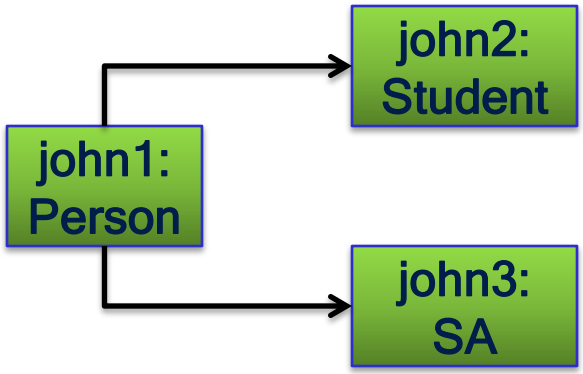


Instance cannot change type at runtime!

Background: Introduction to Roles

John, a student, starts to work as a student assistant.

3. Approach: Delegation

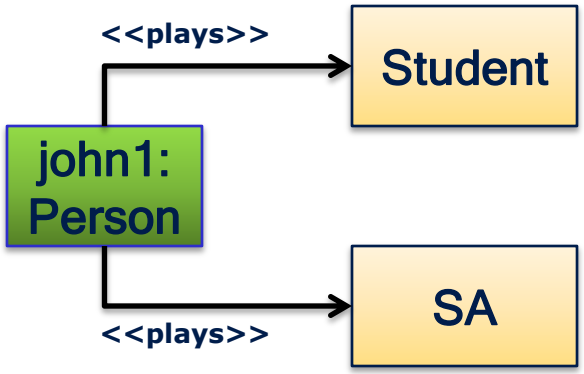


But: conceptual unity → type saftey is lost

Background: Introduction to Roles

John, a student, starts to work as a student assistant.

Roles



Think of dynamic, instance-level aspects.

Background: Introduction to Roles

Other Approaches:

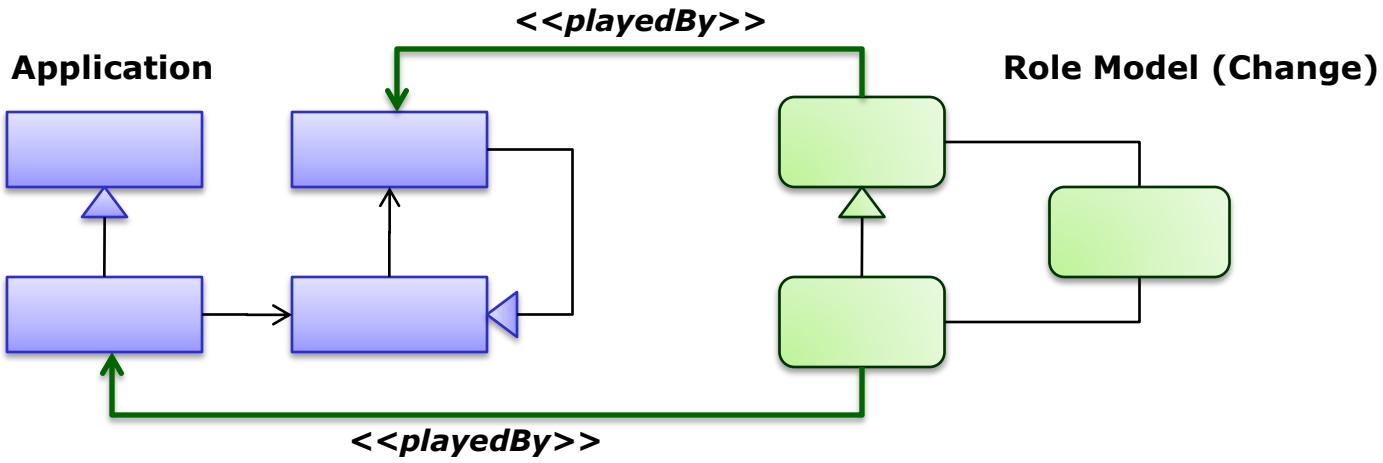
- **Modeling:** ORM (Halpin, 1989), ooRAM SE method (Reenskaug, 1995), …
- More fields, like e.g. Ontologies (Guizzardi, 2005)

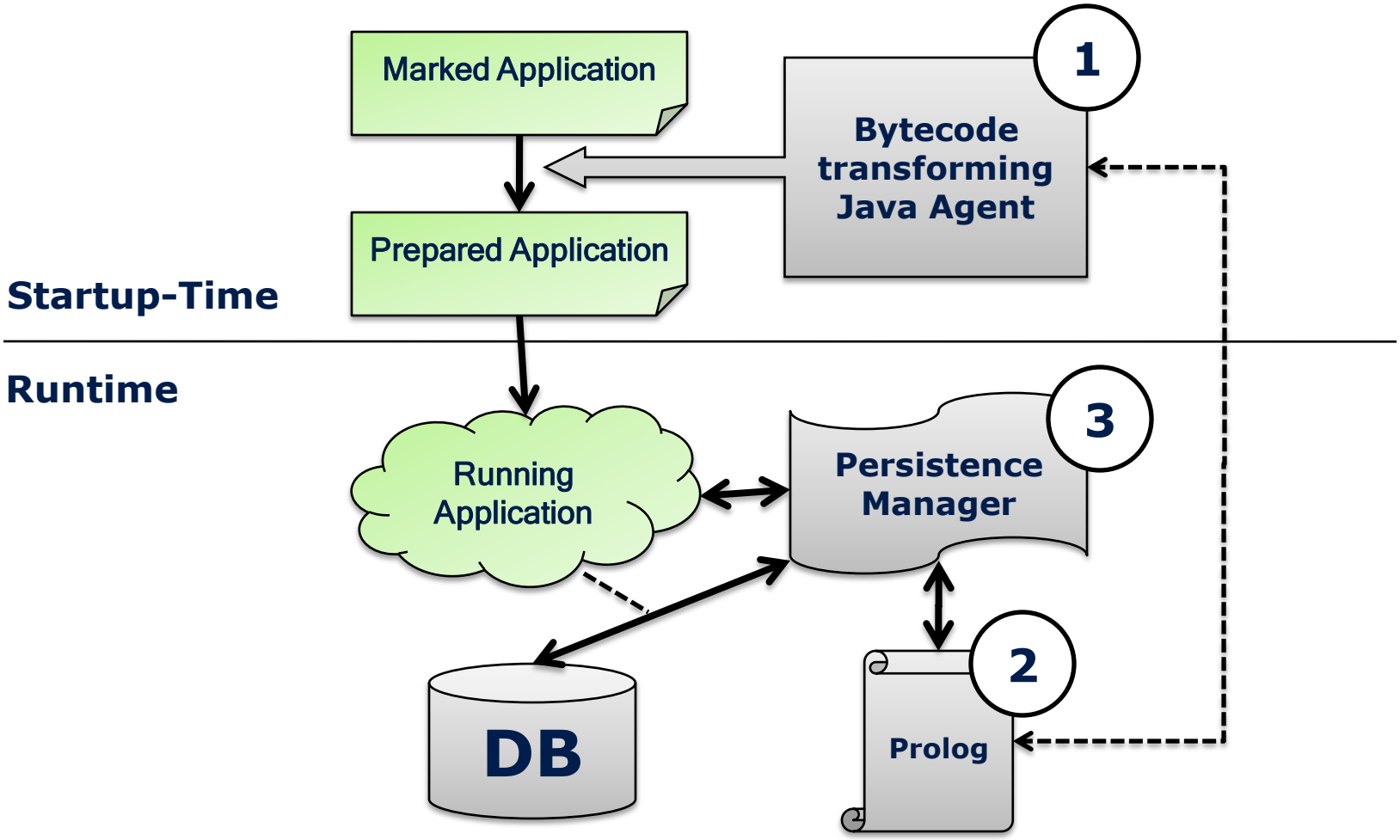
- **Programming Languages**
  - powerJava (Boella et al., 2006)
  - Rava (He et al.,2006)
  - EpsilonJ (Tamai et al., 2007)
  - **ObjectTeams** (Herrmann et al., 2007)

But what about:

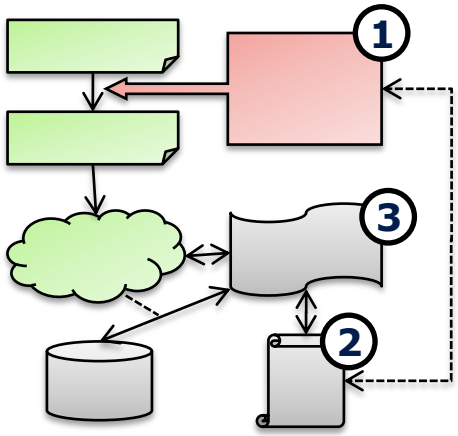
Schema Evolution using Roles

- Only **additions** and **removals** can be derived automatically
  - Developers intend is missing
- But for systems using roles, changes are additions and/or removals
  - Changes are developed as role models



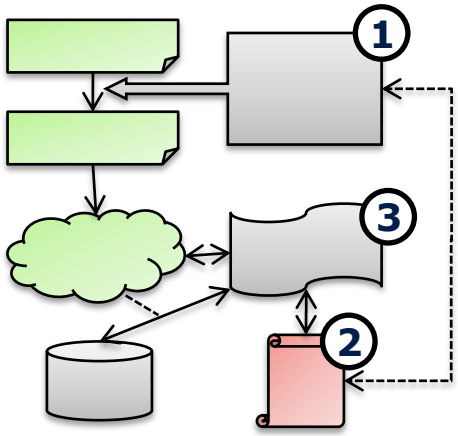

Marked Application
1
Bytecode transforming Java Agent
Prepared Application
Startup-Time
Runtime
Running Application
3
Persistence Manager
DB
2
Prolog

4.1. The Java Agent



- Class loading is intercepted
- Their schema is written into a Prolog fact base (2) or changes are identified by comparison
- Classes and roles are modified
- implicit dataflow and object lifecycle is transformed to an explicit event stream.
  - new objects
  - role playership
  - value changes
- Adjustments to be able to:
  - create,
  - remove and
  - change objects and roles

4.2. Prolog



schema fact base:

```
isClass(´Person´).
hasAttribute(´Person´,´name´,´String´,0).
...
references(´Person´,´partner´,´Person´,_).
...
isRole(´Student´,´Person´,´University´).
subclasses(´Person´,´Mammal´).
```

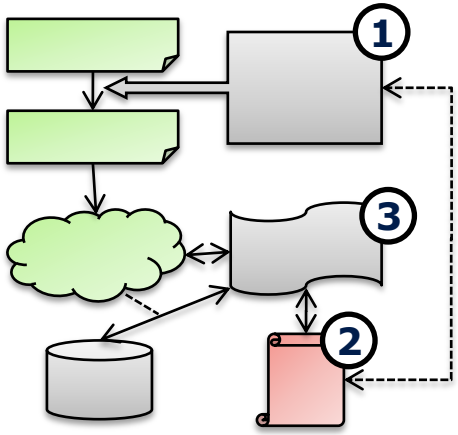
runtime fact base:

```
instanceof(Person,[John,25,...,1]).
instanceof(Person,[Hans,20,...,2]).
instanceof(Person,[Karl,55,...,3]).
```

(For clarity not all predicates are shown)

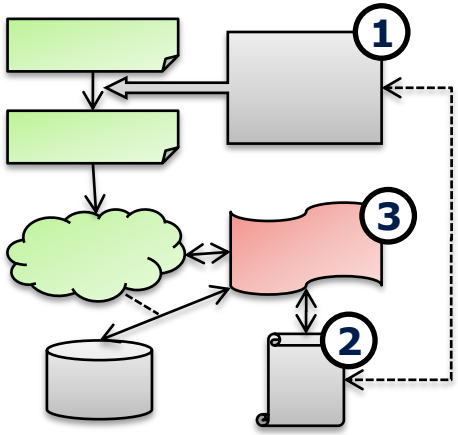
- Prolog instead of **Datalog and F-Logic**, because no mature, production ready and free interpreters exist

4.2. Prolog



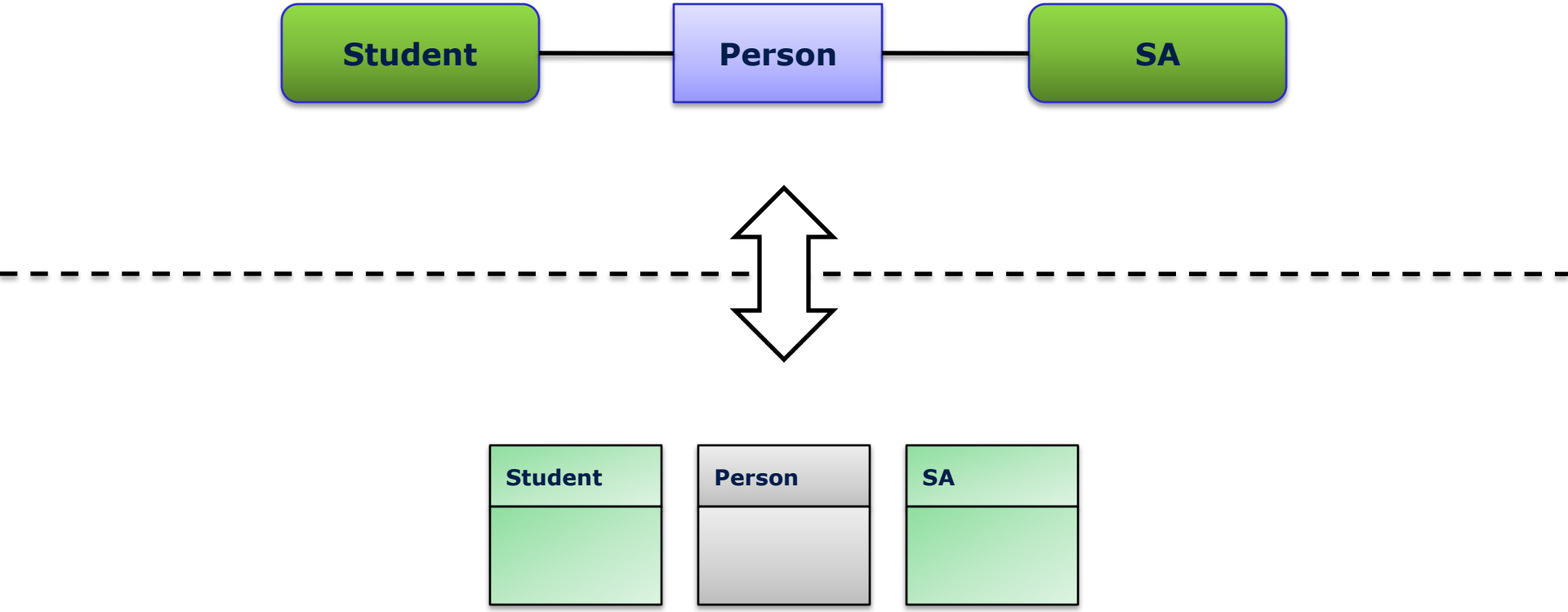
- **Schema fact base**: 19 types of facts
  - isClass/1, isRole/3, isContext/1
  - hasAttribute/4, hasStaticAttribute/4
  - subclasses/2, references/4
  - Remaining fact types related to changes
- **Runtime fact base**: 3 types of facts
  - instanceof/2, sameInstance/4, contextState/3
- **Predicates to derive** further information from facts
  - 108 predicates
  - e.g. update instances to new schema
- **Transparency**
- **Reuse** of knowledge possible
  - Analysis, Optimizations
  - Transformations (e.g. normalization)

4.2. Persistence Manager



- Traces the applications events into Prolog
- Offers search criteria API + restore functionality
- Triggers persistence mechanism
  - Object-Relational mapping well known
  - But Role-Relational mapping not!

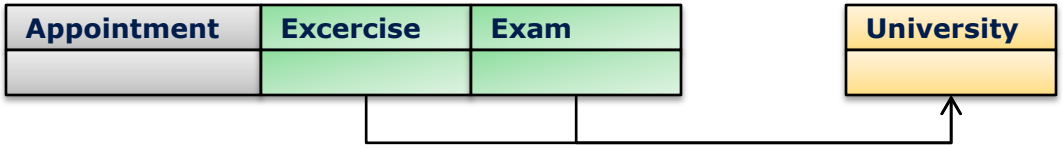
Object-Role-Relational Mapping



Role-Relational Mapping:

Class-Role Relations

- High schema and data redundancy

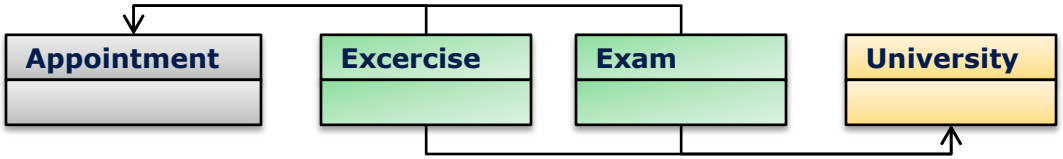


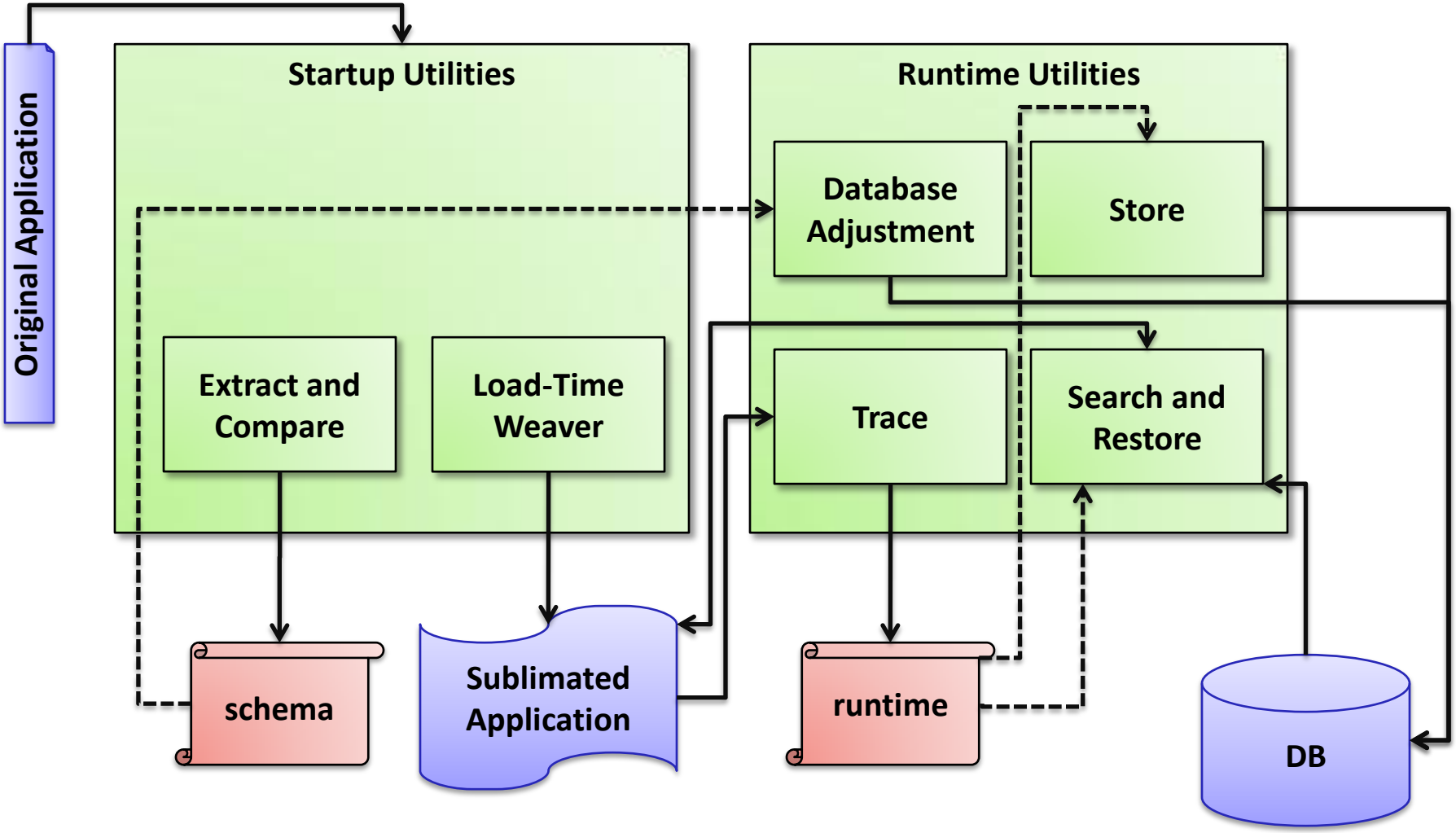
Normalized Class-Role Rel.

- Avoids schema and data redundancy by normalizing the schemata
- Normalization revealed to be too time consuming (>1min for a relation with 20+ attributes)

Complete Separation

- Role types have two additional attributes:
  1. current player
  2. current context




Original Application
Startup Utilities
Runtime Utilities
Database Adjustment
Store
Extract and Compare
Load-Time Weaver
Trace
Search and Restore
schema
Sublimated Application
runtime
DB

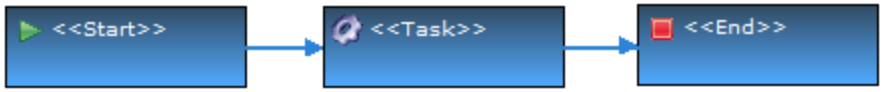
- **Javassist** for bytecode transformations

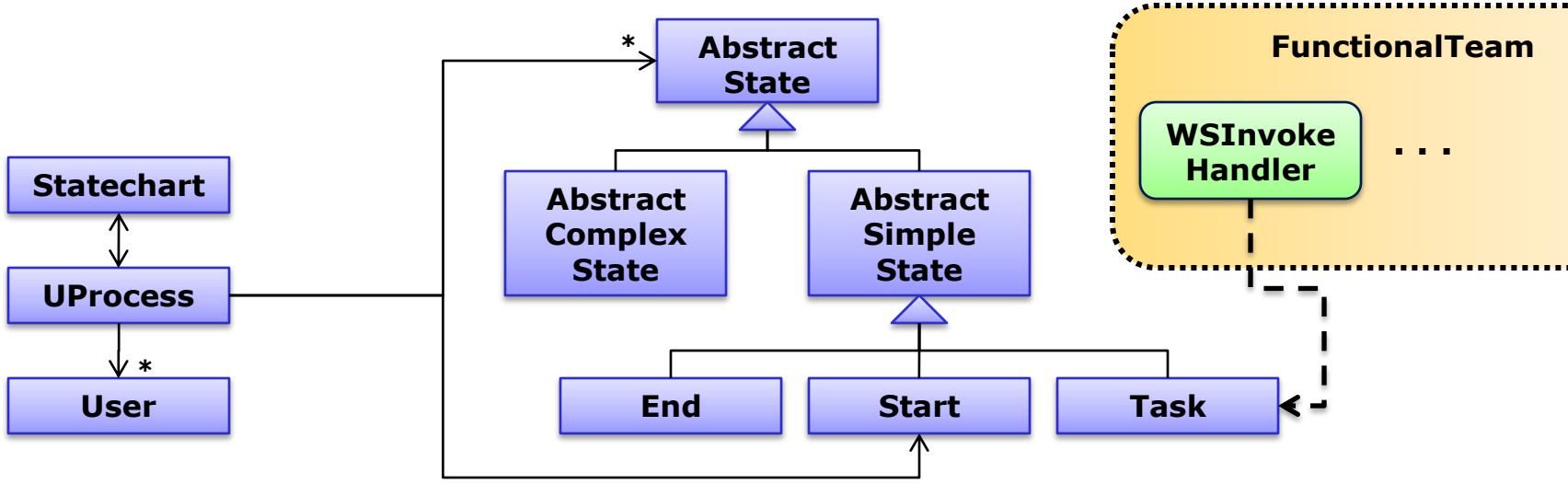
- **Java™ agent** (java.lang.instrument) for load-time weaving

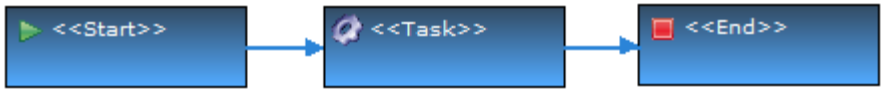
- SWI Prolog / **JPL** (Java Prolog API)

4 KLOC Java™	400 LOC Prolog

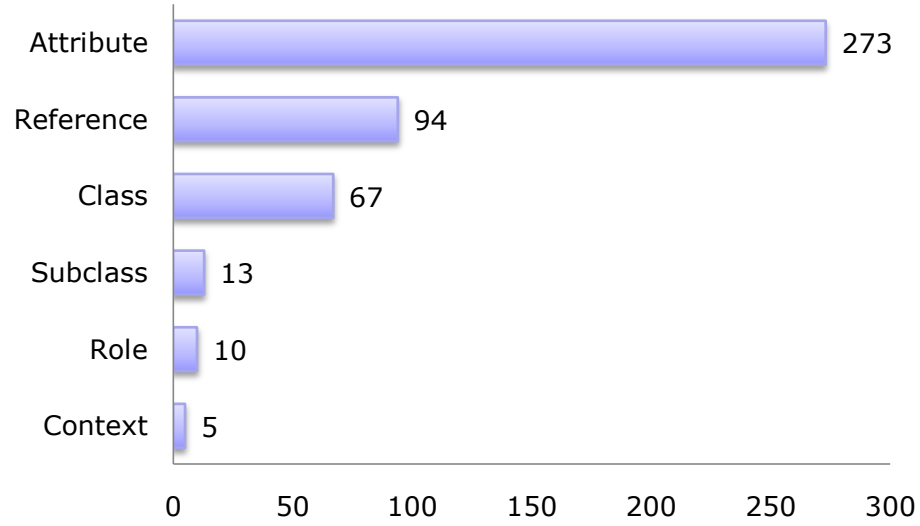
- **integrated into OSPP** (~80 KLOC), a multi-purpose workflow engine
- simplest setup (a single trivial workflow)



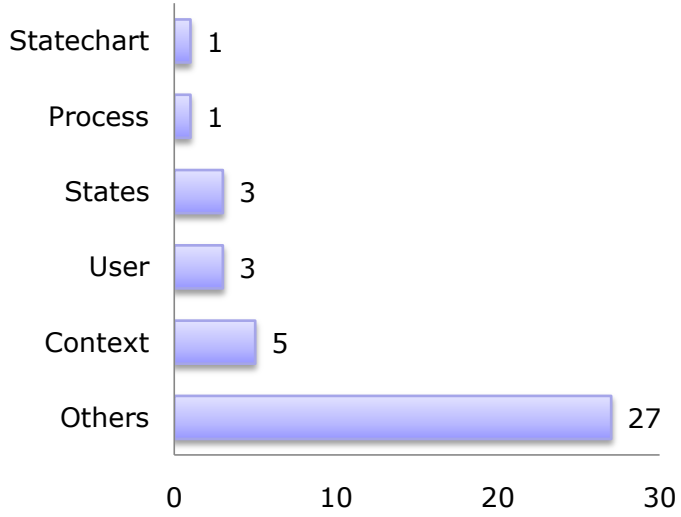




480 facts in **schema fact base**

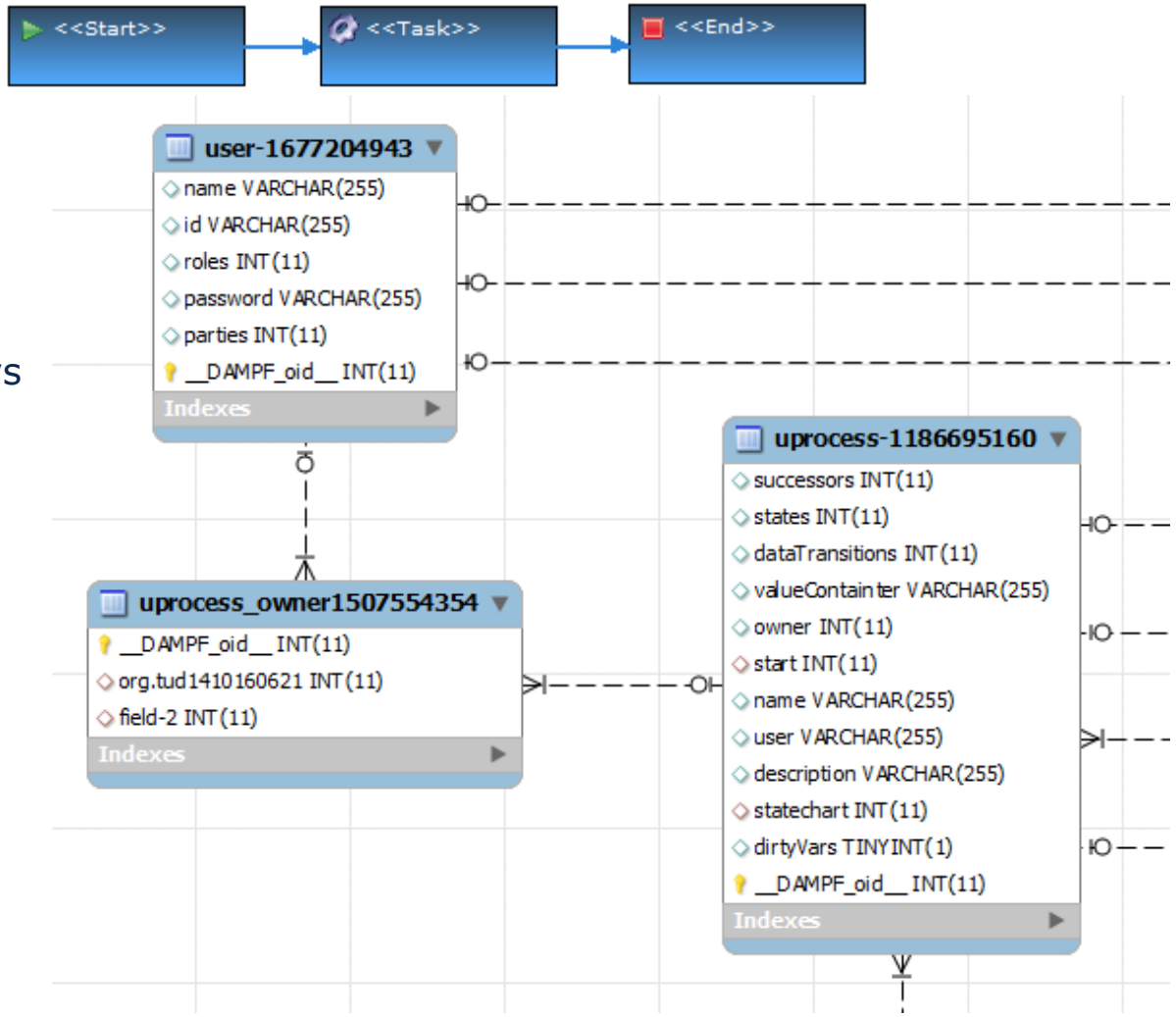


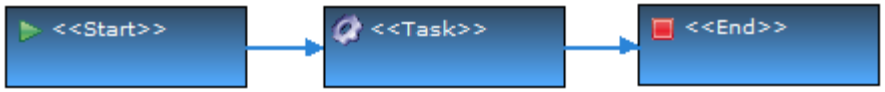
39 facts in **runtime fact base**

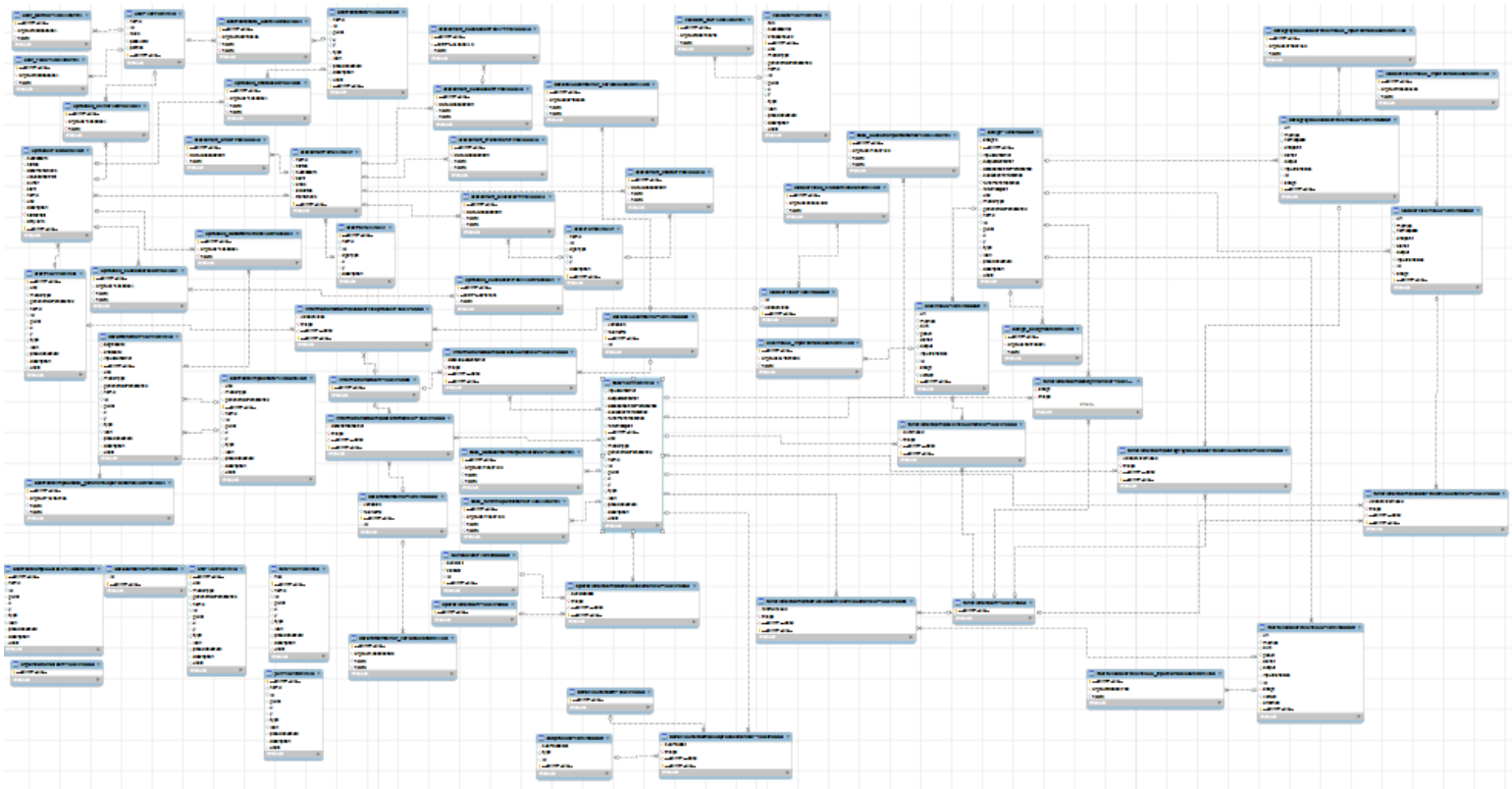


Database

- **67** relations
- **72** foreign keys
- takes **~7s** to create

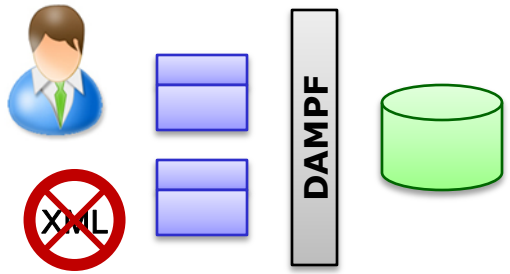



<<Start>>
<<Task>>
<<End>>

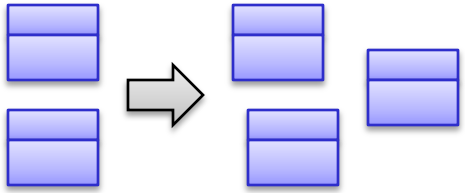


The Features of DAMPF:

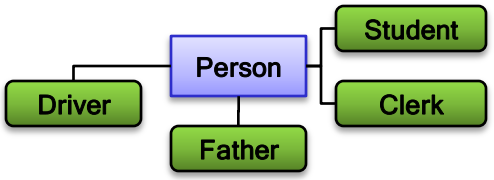
1. Transparency



2. Support for Schema Evolution



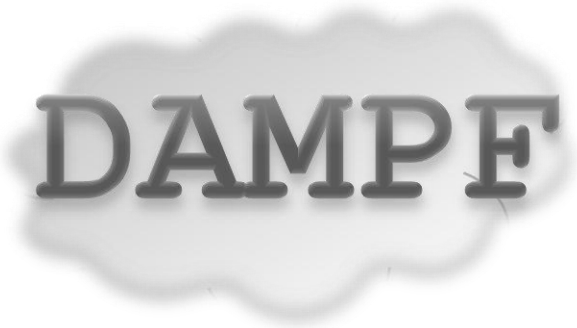
3. Support for Roles



07 Future Work

- Improve the prototype
  - Transaction Support
  - Mass data support
  - Automatic / Derived Optimization

- Provide publicly
- Publish (planned for ICOODB 2010)
- Discover new application areas



Questions ?

Schema Evolution – how far can we go?

Addings / Removals

➔ can be automatically detected

Intra-Class Refactorings / Changes

➔ E.g. RenameAttribute, but also ExtractSubclass

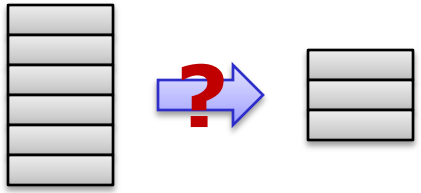
➔ intend of developer required
(ComeBack!, RefactoringCrawler, …)

Inter-Class Refactorings / Changes

➔ E.g. MoveAttribute

➔ value-mapping information missing

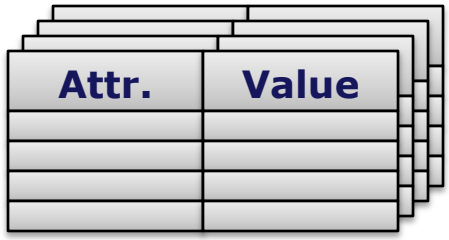
➔ Refactorings are Class-Level



Mapping static attributes to relations

Two approaches

- ➔ **Single relation** pointing to the classes
  - ➔ classname, attributename, value
- ➔ Seperate relations
  - ➔ For each class an additional relation is added
  - ➔ Choice:
    - ➔ **one column per static attribute** or
    - ➔ **Two columns**: attributename, value



Java Persistance API

```
@Entity
public class Student {

   private int id;
   private String name;
   private CourseOfStudy cos;

   @Id @GeneratedValue
   public int getId() {return id;}
   public int setId(int id) {
         this.id=id;
   }

   //getter/setter for name

   @ManyToOne
   @JoinColumn(name=´COS_ID´)
   public CourseOfStudy getCOS() {
         return cos;
   }
}
```

VS.

DAMPF

```
@Entity
public class Student {


   private String name;
   private CourseOfStudy cos;

   //getter/setter for name


public CourseOfStudy getCOS() {
         return cos;
   }
}
```

```
public class StudentMgr {
   private Set<Student> students;
   ...
   public void immatriculate(int matrikel) {
      students.add(new Student(matrikel));
   }

   @Remove
   public void removeStudent(Student s) {
         students.remove(s);
   }

   public Student getStudent(int matrikel) {
      PersistenceManager pm;
      Map<String,String> criteria = new HashMap<String,String>();
      criteria.add(„matrikel“,“3013737“);
      Object o = pm.getObject(Student.getClass().getName(),criteria);
      return (Student)o;
   }
}
```

traced, persistence depends on chosen persistence strategy

remove code woven at end of method

automatically injected

Schema Factbase Predicates:

- isClass(´Person´).
- isContext(´University´).
- isRole(´Student´,´University´,´Person´).
- hasAttribute(´Student´,´matrikel´,´int´,0).
- hasStaticAttribute/4
- references(´Student´,´lectures´,´Lecture´,_).
- subclasses(´Person´,´Mammal´).

- AddedClass, RemovedClass, AddedRole, RemovedRole, AddedContext, RemovedContext
- AttachedSuperclass, DetachedSuperclass
- AddedAttribute, RemovedAttribute, ChangedAttribute (only position)
- ChangePlayer
- AddedReference, RemoveReference (not facts, but predicates)

Runtime Factbase Predicates:

- instanceof(´Person´,[´john´,1,2]).
- sameInstance(´Person´,´Mamal´,2,4).
- contextState(´University´,3,true).

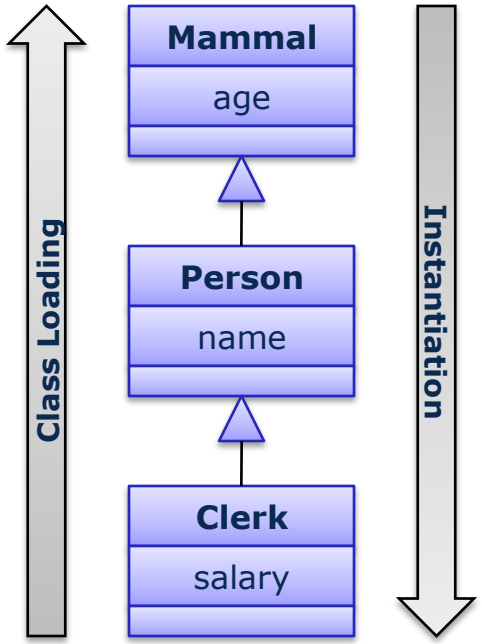
Inheritance

new Clerk(„John“,25,1199.95);

1. instanceof(´Mammal´,[´25´,1]).
   sameInstance(´Mammal´,´Clerk´,1,-1).

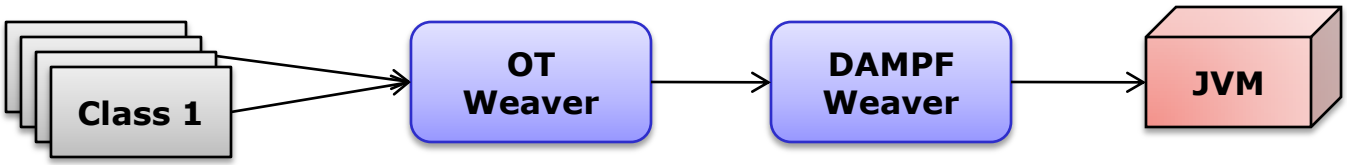
2. instanceof(´Mammal´,[´25´,1]).
   instanceof(´Person´,[´John´,2]).
   sameInstance(´Mammal´,´Clerk´,1,-1).
   sameInstance(´Person´,´Clerk´,2,-1).

3. instanceof(´Mammal´,[´25´,1]).
   instanceof(´Person´,[´John´,2]).
   instanceof(´Clerk´,[1199.95,3]).
   sameInstance(´Mammal´,´Clerk´,1,3).
   sameInstance(´Person´,´Clerk´,2,3).



Load-Time Weaving vs. Post-processing

- problem with other Load-Time Weavers (OT)



Support for Reflection?

- yes, except sun.misc.Unsafe

What about debugging modified code?

- it's a problem
- Eli Tilevich (Virginia Tech) is working on an approach for it

How much effort is it, to introduce DAMPF in existing systems

- it depends
  - existing persistence mechanism needs to be removed (takes much time)
  - all domain classes need to be annotated

How to support a new role language

- Java™ Agent needs to be adjusted, due to different implementation of roles
- all other parts do not need to be touched

How much is the footprint?

- DAMPF 100kb + JPL 128kb + Javassist 600kb + JDBC driver (MySQL ~700kb)
  → **1.5MB**
- additionally Prolog: 22MB

08 Backup Slides

How do you map inheritance to the relational schema?

- Joinable Table Per Class
  (each class get's its own relation, with only its fields)
- Optimization possible (future work)

How do you handle polymorphism?

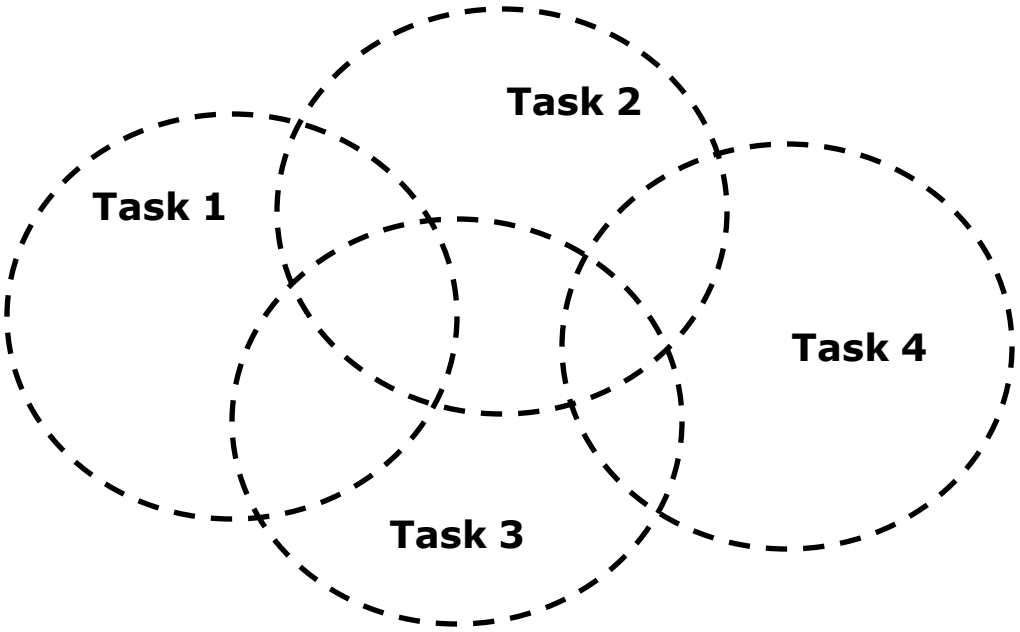
- classes may reference superclasses, interfaces or abstract classes
  → how to handle the foreign keys?
- instances are inserted partially into "super"-relations
- tradeoff: data redundancy vs. data integrity

Do you support nested classes?

- yes, btw. roles are realized in ObjectTeams as nested classes

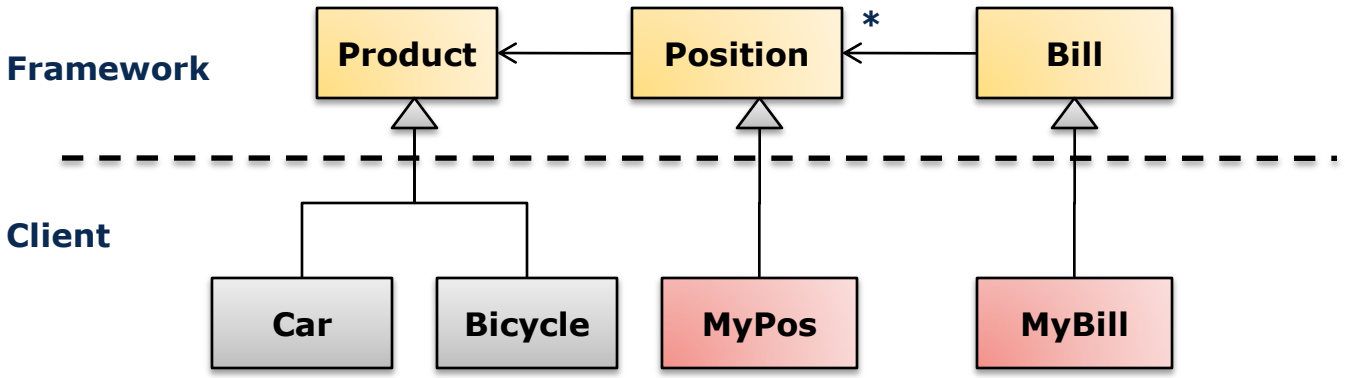
What are Roles good for, except dynamic type composition?

- better maintainability + cleaner structure in regard to behavior
  - tasks of the system are encapsulated by role models
  - instead of being scattered over and tangled in the classes



Does using frameworks pose a problem?

- domain model needs to be completely in the client



- MyPos could be avoided by

```
@Entity(followReferences=true)
public class MyBill extends Bill
```