JDBC vs iBATIS

A case study

Fabrizio Gianneschi

Atlantis S.p.A.

fabrizio.gianneschi@gruppoatlantis.com

About iBATIS

- Open source framework that reduces the complexity to read/save Java objects from a database (*persistence*)
- Decouples Java code from SQL
- Avoids the necessity to use JDBC
- It is NOT an object-relational mapper (such as *Hibernate*)
- Simplifies the developer's life ☺

Source: http://www.ibatis.com

10 Reasons to use iBatis

1. Works with any database that has a JDBC driver (no plugins required)
2. Configurable caching (including dependencies)
3. Local and Global transaction support and management (JTA)
4. Simple XML mapping document structure
5. Supports Map, Collection, List and Primitive Wrappers (Integer, String etc.)

Source: http://www.ibatis.com

10 Reasons to use iBatis

6. Supports JavaBeans classes (get/set methods)
7. Supports complex object mappings (populating lists, complex object models etc.)
8. Object models are never perfect (no changes required!)
9. Database designs are never perfect (no changes required!)
10. You already know SQL, why waste time learning something else?

Components

iBATIS is divided into two main components:

- **SQLMaps**
  Permits to read/save Java objects into relational DBMS without using JDBC and without mixing Java and SQL code
- **DAO**
  “Is an abstraction layer that hides the details of your persistence solution and provides a common API to the rest of your application” (src: iBATIS)

SQLMaps

SQLMaps in pills

- Based on XML configuration files; you have to give a *name* (or an *id*) to each query used in the application
- Query input parameters could be primitives or simple classes (Integer, String…) or HashMap
- In complex cases, application classes could be used directly (es. VO, DTO)
- Particular DB situations are handled using column / property mappings
- The same principles are also valid for output parameters

Simple Mapping

```
<?xml version="1.0" encoding="UTF-8" ?>

…

<sqlMap namespace="Persona">

    <select id="findPersonaById" resultClass =
          "com.gruppoatlantis.www.verbali.dto.PersonaDTO">

        SELECT *
        FROM persone_view
        WHERE idpersona=#value#

    </select>

…
```

```
<?xml version="1.0" encoding="UTF-8" ?>

…

<sqlMap namespace="Persona">

  <resultMap id="personaMap"
      class="com.gruppoatlantis.www.verbali.dto.PersonaDTO">

    <result property="id" column="idpersona"/>
    <result property="cognome" column="cognome"/>
    <result property="nome" column="nome"/>
    <result property="matricola" column="matricola"/>
    <result property="livello" column="eta" jdbcType="INTEGER"
            nullValue="0" />
  </resultMap>

    <select id="findPersonaById" resultMap="personaMap">

       SELECT *
       FROM persone_view
       WHERE idpersona=#value#

    </select>

…
```

Complex Mapping

SQLMaps in pills

- To execute a query within Java classes, you have to use an instance of the class **`com.ibatis.sqlmap.client.SqlMapClient`**
- The most used methods are

  `queryForList` and **`queryForObject`** on which you pass the name of query and an object as a parameters
  As an example:

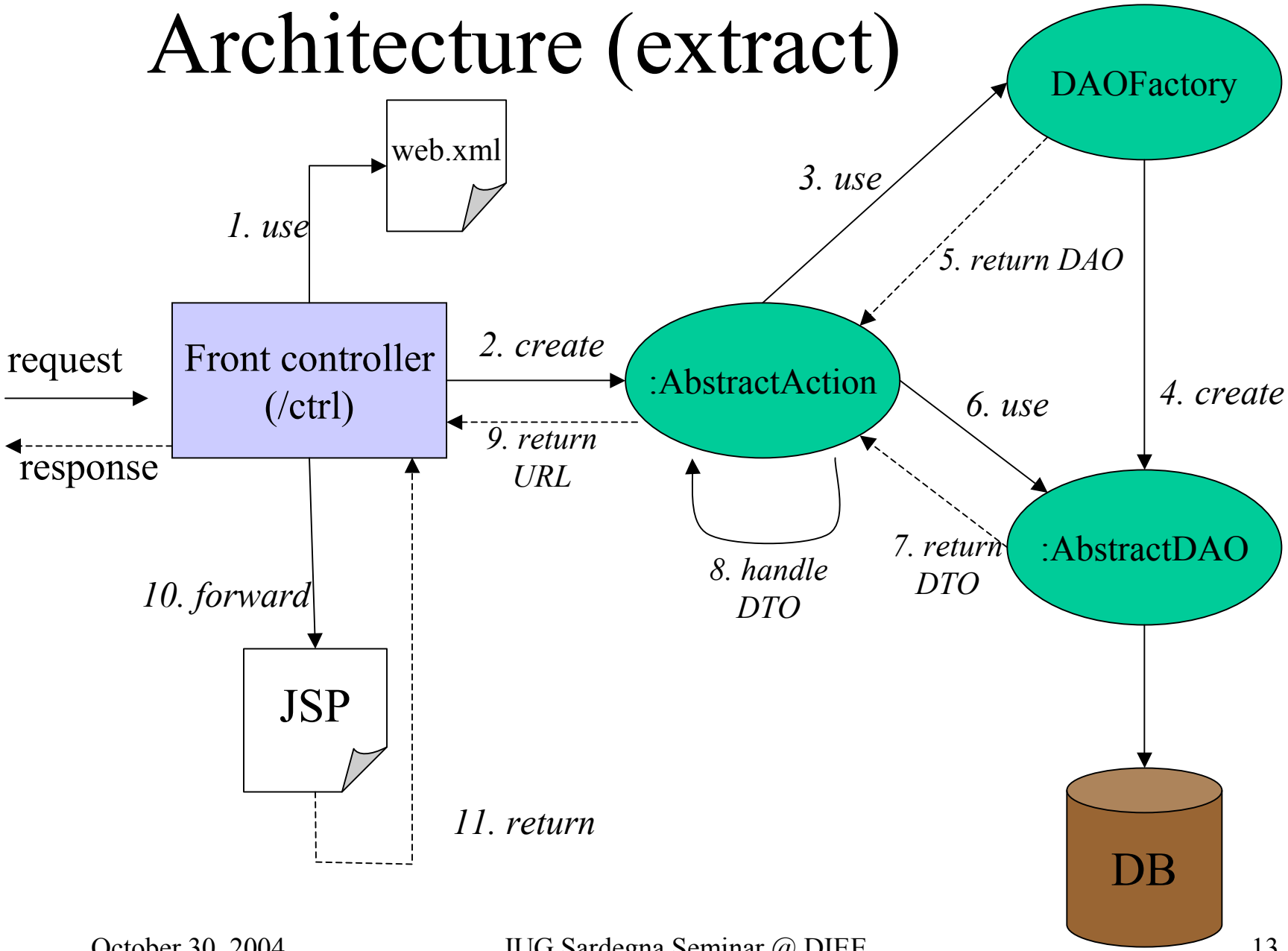
```
SQLMapClient sqlMap = …;
String queryName = "findPersonaById";
Integer id = new Integer(15);
PersonaDTO p = (PersonaDTO) sqlMap.queryForObject(queryName, id);
```

SQLMaps case study

Scenario

- Simple Intranet application based on **J2EE** technology (JSP, JSTL, Servlet), in production since January 2004, that process and mantains the reports of the corporate's internal meetings.
- Data access through DAO/**JDBC**
- PostgreSQL database (14 tables, 3 views, 9 sequences, 1 function)

Architecture (extract)



LOC count before iBATIS*

AbstractDAO	26
AttivitaPostgresDAO	524
CategoriaPostgresDAO	151
MascheraDAO	221
OdgPostgresDAO	131
PersonaPostgresDAO	388
ProgettoPostgresDAO	112
RiunionePostgresDAO	514
RuoloPostgresDAO	75
StatoAttivitaPostgresDAO	124
TOTAL	**2266**

* Only DAO Layer

JDBC Java method (1/3)

```
public List findXyZ(String par1) {
   Connection conn = null;
   List retColl = null;
   try {
      conn = getConnection(); //implemented on superclass
      String query = "SELECT * FROM table WHERE a=?;";

      PreparedStatement ps = conn.prepareStatement(
          query, ResultSet.TYPE_SCROLL_INSENSITIVE,
          ResultSet.CONCUR_READ_ONLY);

      ps.setString(1 , par1); // sets query parameters

      ResultSet rs = ps.executeQuery();
//(... continue ...)
```

JDBC Java method (2/3)

```
retColl = new ArrayList(); // result collection

while ( rs.next() ) { // iterate over query results

    // an object for each row
    MyClass obj = new MyClass();
    obj.setField1( rs.getInt(1) );
    obj.setField2( rs.getDate(2) );
    …
    obj.setFieldN( …. );
    retColl.add(obj);
}
rs.close(); // release resources
ps.close();
return retColl;
//(... continuea ...)
```

JDBC Java method (3/3)

```
        //…

        } catch (SQLException ex) {
           // error handling
           …
        } finally {
            // implemented on superclass
            closeConnection(conn);
        }

} // End of method
```

iBATIS Java method

```
public List findXyZ(String par1) {
    try{

        return sqlMap.queryForList("findXyZ",par1);

    }catch(SQLException ex){
       // error handling
       ...
    }
} // End of method
```

JDBC-iBATIS Comparison

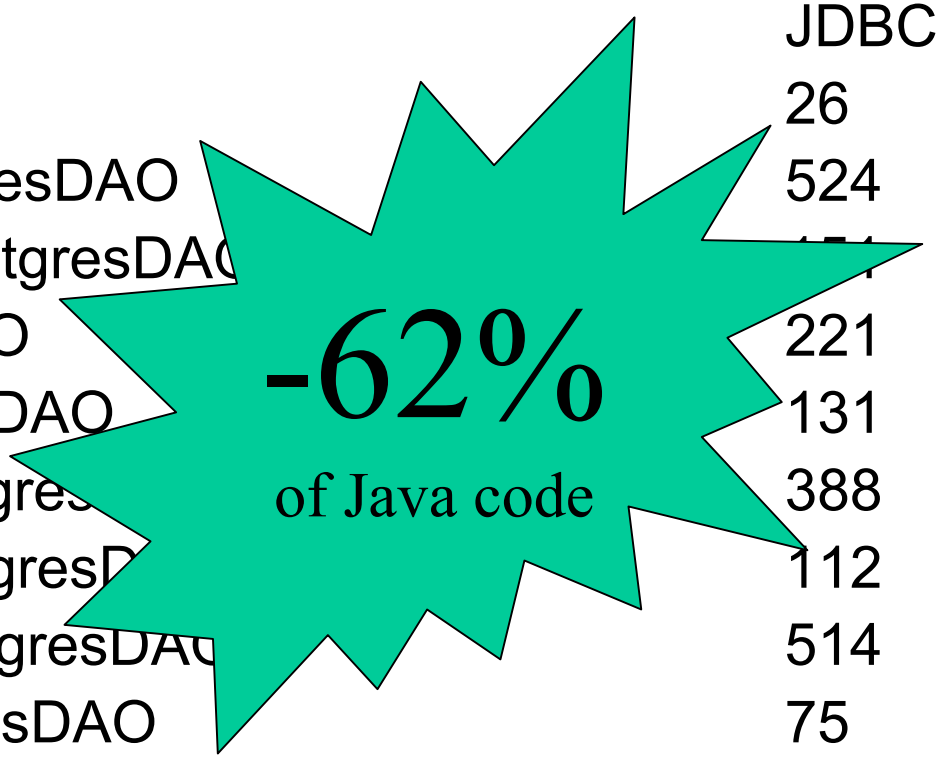
JDBC

1. Obtain the db Connection
2. Create the statement
3. Set the input parameters
4. Execute the statement
5. Create the result Collection
6. For each row fetched:
   a) Create an object
   b) Set object's properties using row's colums
   c) Add the object to the Collection
7. Release resources
8. Close the Connection
9. Return the Collection

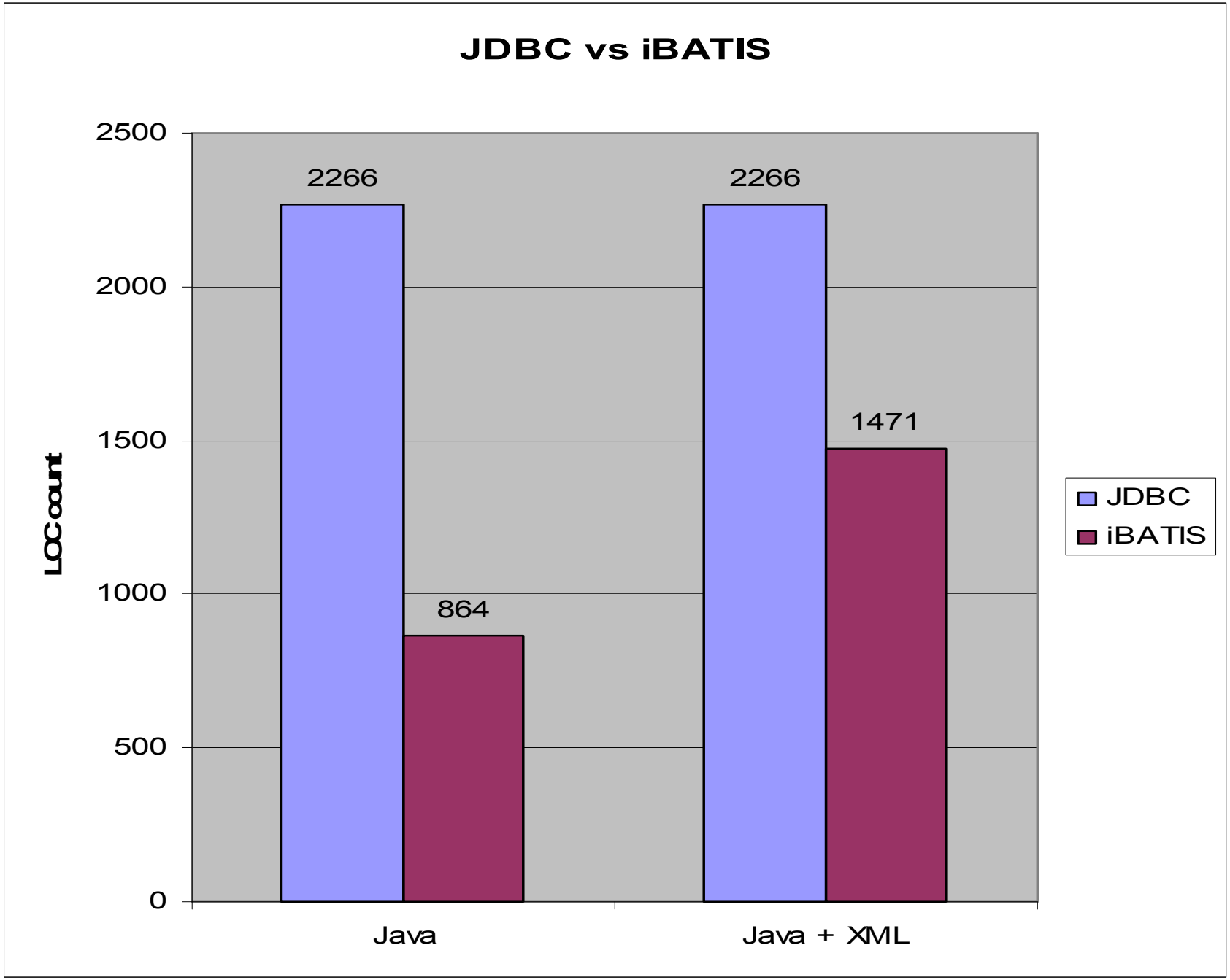
iBATIS

1. Obtain the SqlMap object
   a) Prepare complex parameters (optional)
2. Invoke the query passing the parameters
3. Return the result

LOC count



	JDBC	iBATIS
AbstractDAO	26	0
AttivitaPostgresDAO	524	199
CategoriaPostgresDAO	[illegible]	66
MascheraDAO	221	73
OdgPostgresDAO	131	51
PersonaPostgres[illegible]	388	165
ProgettoPostgres[illegible]	112	26
RiunionePostgresDAO	514	202
RuoloPostgresDAO	75	34
StatoAttivitaPostgresDAO	124	48
TOTALE	**2266**	**864**


JDBC vs iBATIS
LOC count
2500
2000
1500
1000
500
0
2266
864
2266
1471
Java
Java + XML
JDBC
iBATIS

Performance

- Comparable with the “classic” approach enabling some options
  (http://raibledesigns.com/page/rd?anchor=persistence_options_with_existing_sql)
- Advanced Object Caching
- Lazy loading
- Stored procedures support
- The SQL code is visible and accessible for optimization

Conclusions

- The introduction of iBATIS into the project has been extremely favorable in terms of LOC count reduction and general clarity
- The business logic and the old SQL queries were not modified.
- The learning curve was very good (< 1 day for the first working DAO)

iBATIS – Hibernate comparison

	iBATIS	Hibernate
Suitable since the beginning of the project		
Suitable in the middle/end of the project		
OR mapping		
Simplicity		
Features		
Documentation		
Community		

References

Official references:

- http://www.ibatis.com
- http://www.ibatis.com/common/sqlmaps.html
- http://opensource.thoughtworks.com/projects/ibatis.jsp

Articles:

- http://www.sixty4bit.com/mt/archives/2003/12/17/persistence_layer_review.html
- http://www.developer.com/db/article.php/3346301