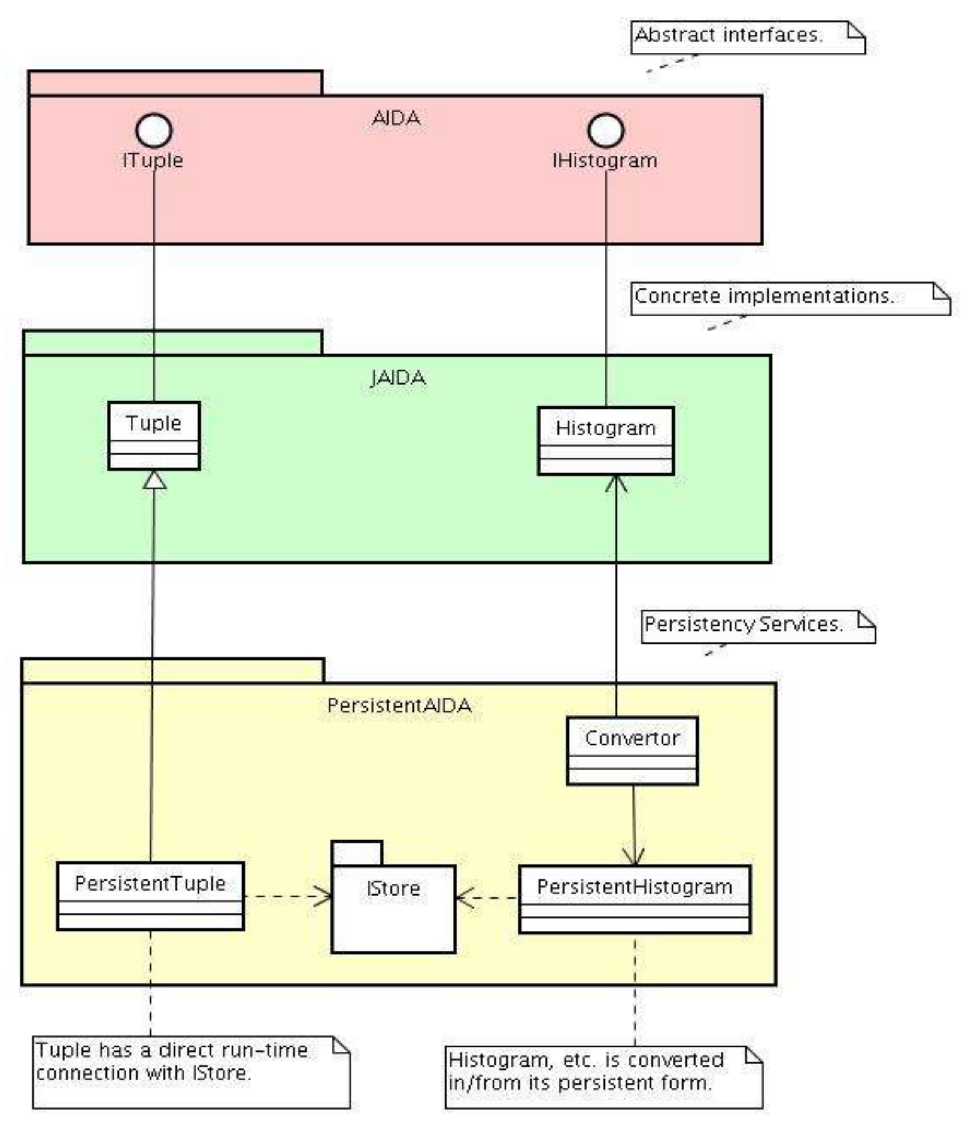
PERSISTENCE FOR ANALYSIS OBJECTS

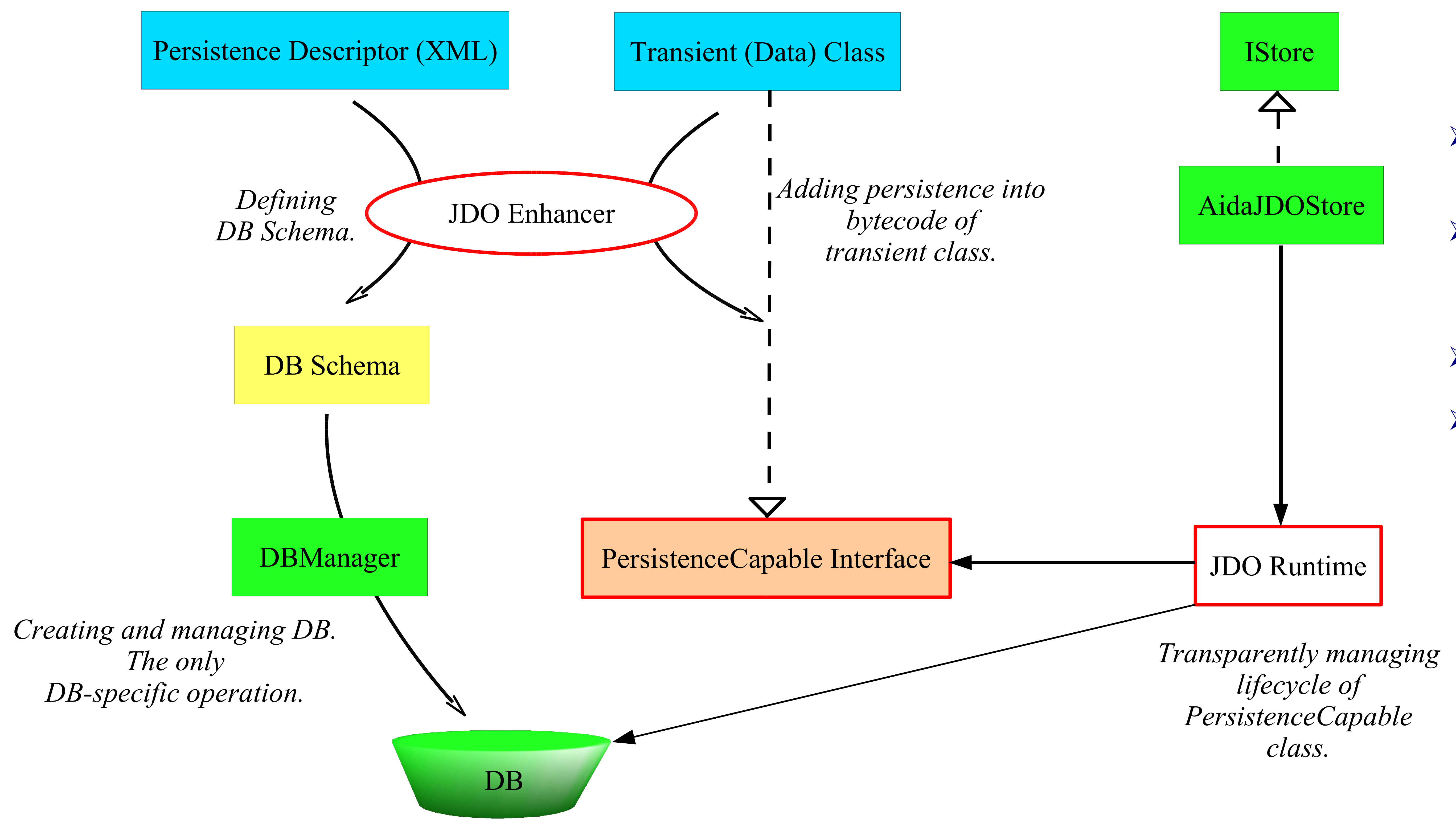
JULIUS HRIVNAC - LAL/ORSAY



There are two kinds of AIDA objects with respect to persistence requirements:

- ➢Objects, which are read/written entirely in one step: IHistograms, IClouds, IProfiles, ... All Persistence requirements for those objects can be implemented by standard persistence techniques based on Transient-Persistent Separation (JDO, Serialization,...)
- ➢Objects, which are read/written only partially in one step and objects which are only interrogated: ITuples. It is not feasible to completely separate Transient and Persistent form of those objects. Their Persistence should be tightly interfaced with their transient form. One possibility (chosen here for SQL-based persistence) is to implement an ITuple extension for each persistence mechanism.

ORTHOGONAL PERSISTENCE


Persistence Descriptor (XML)
Transient (Data) Class
IStore
Defining DB Schema.
JDO Enhancer
Adding persistence into bytecode of transient class.
AidaJDOStore
DB Schema
DBManager
PersistenceCapable Interface
JDO Runtime
Creating and managing DB. The only DB-specific operation.
Transparently managing lifecycle of PersistenceCapable class.
DB

- ➢JDO is currently the mainstream persistence technology for Objects.
- ➢JDO implementations (both free and commercial) exist for practically all existing Databases (relational or not).
- ➢All JDO Enhancers are compatible.
- ➢JDO provides all standard database functionality (transaction, caching,...).

FreeHEP AIDA can be persistified using **Java Data Objects (JDO)**.

- ➢FreeHEP AIDA storage API (IStore) should be enhanced to support real database (the current one supports only file-like API).
- ➢AIDA objects can be decoupled from ITree management.
- ➢More powerful query machinery can be introduced.

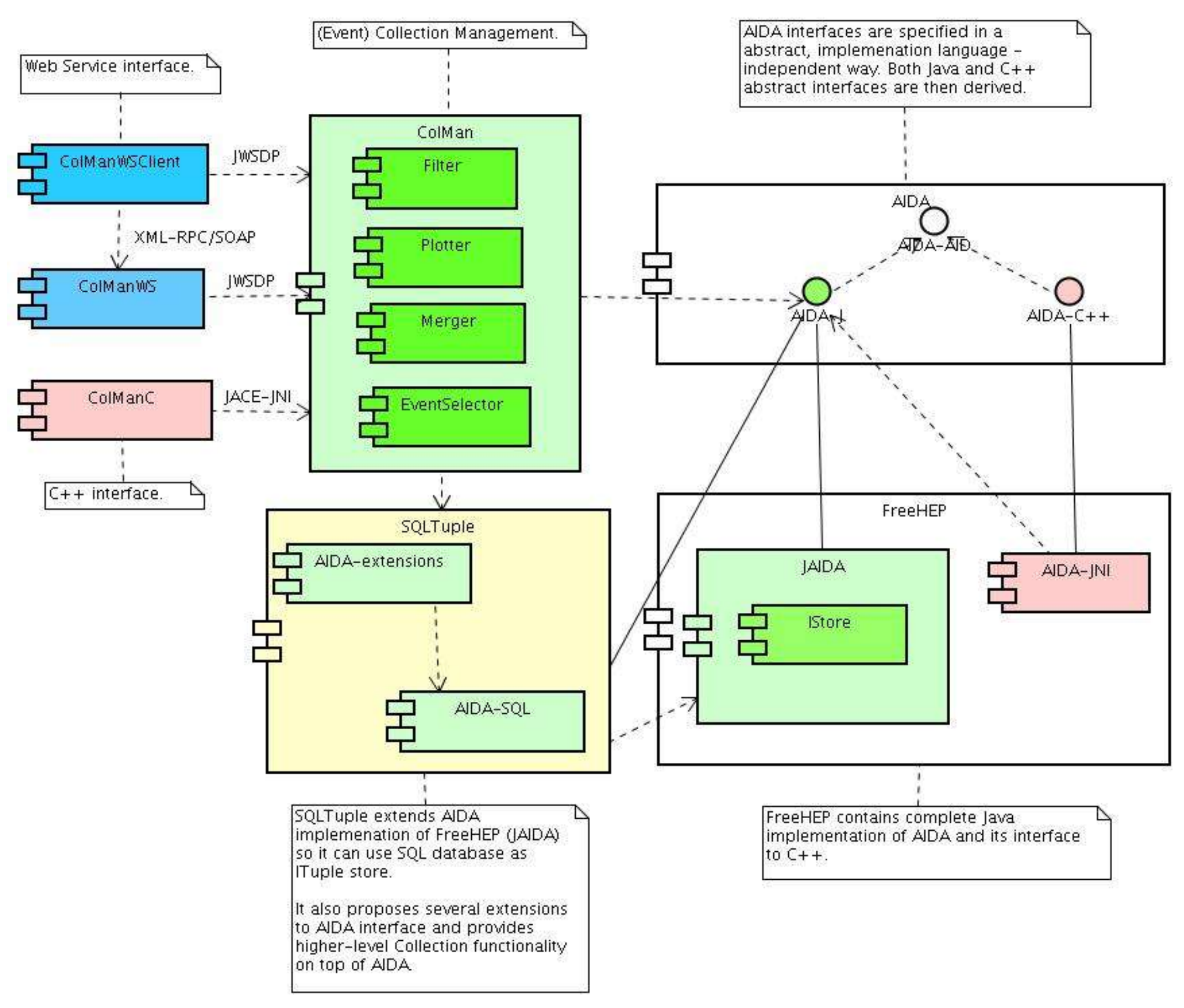
AIDA specifies Interfaces, but persistence should work on Data. There are two ways to handle it:

- ➢Persistify current FreeHEP implementation of AIDA.
  - ➢Easy to implement.
  - ➢Faster.
  - ➢Not portable (depends on actual FreeHEP implementation).
- ➢Create AIDA DataClasses (from AIDA XML Schema by XSLT StyleSheets) and persistify them.
  - ➢Transient-Persistent converters should be written to convert between DataClasses and actual implementation of AIDA. Non-intrusive Aspects can be used as converters.
  - ➢Slower.
  - ➢Portable (AIDA XML Schema is common to all AIDA implementations).

```
/** Plot all 1-dim histograms from a database. */

// Start AIDA
IAnalysisFactory af = IAnalysisFactory.create();
IPlotter page = af.createPlotterFactory().create("Page");
page.show();

// Start JDO
PersistenceManager pm = Connection.getPM(databaseProperties);
Transaction tx = pm.currentTransaction();
tx.begin();

// Get Histograms
Query query = pm.newQuery(Histogram1D.class);
Collection result = (Collection)query.execute();
page.createRegions(result.size() / 2 + 1, 2);

// Use Histograms
Iterator it = result.iterator();
int i = 0;
while (it.hasNext()) {
  page.region(i++).plot((IHistogram1D)it.next());
  }

// Close JDO
tx.commit();
pm.close();
```

DIRECT PERSISTENCE

http://home.cern.ch/hrivnac/Activities/packages/SQLTuple



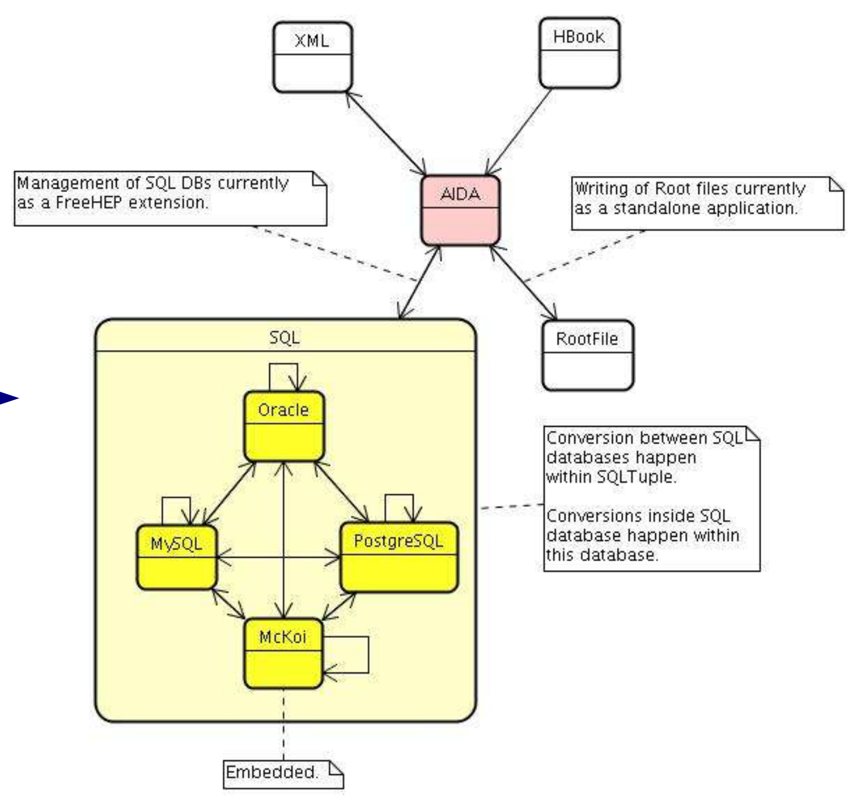
Web Service interface.
(Event) Collection Management.
AIDA interfaces are specified in a abstract, implemenation language - independent way. Both Java and C++ abstract interfaces are then derived.
ColManWSClient
JWSDP
XML-RPC/SOAP
ColManWS
JWSDP
ColManC
JACE-JNI
C++ interface.
ColMan
Filter
Plotter
Merger
EventSelector
AIDA
AIDA-AID
AIDA-J
AIDA-C++
SQLTuple
AIDA-extensions
AIDA-SQL
FreeHEP
JAIDA
IStore
AIDA-JNI
SQLTuple extends AIDA implemenation of FreeHEP (JAIDA) so it can use SQL database as ITuple store.
It also proposes several extensions to AIDA interface and provides higher-level Collection functionality on top of AIDA.
FreeHEP contains complete Java implementation of AIDA and its interface to C++.

SQLTuple implements AIDA interface as an extension of the FreeHEP JAIDA implementation. ColMan contains several extensions (Filter, Plotter, Merger, EventSelector, Replicator) using AIDA to perform global operations on NTuples (not only SQLTuples). All the functionality is available using standard AIDA interfaces, WebService access and JACE-created C++ proxies.

SQLTuple implementation of AIDA interface (including its extensions) is in *hep.aida.ref.sql* package. Higher level extensions are in *net.hep.atlas.Database.Collections.Management* package.

AIDA NTuples can be stored using many different storage technologies (Compressed XML files, Root files, HBook files, several SQL databases).

Operations between technologies (filtering, merging,...) are possible via standard AIDA channels. Within SQL technology, native SQL channels are used to speed up operations.



XML
HBook
AIDA
Management of SQL DBs currently as a FreeHEP extension.
Writing of Root files currently as a standalone application.
RootFile
SQL
Oracle
MySQL
PostgreSQL
McKoi
Conversion between SQL databases happen within SQLTuple.
Conversions inside SQL database happen within this database.
Embedded.

Advantages over alternative Tools:

- SQLTuple runs on any platform without recompilation and can be compiled on any platform using any Java compiler version 1.5+. Distribution compatible with Java 1.4 can be provided if needed.
- All Relational Databases can be supported in local (embedded) or remote (server) mode (as long as such modes are supported by the database).
- The performance of SQLTuple is in most cases higher than performance of equivalent C++ implementation.
- SQLTuple can be easily used from other languages, like Python, Ruby, Groovy, PNuts or C++ or as a language-neutral Web-Service.
- All SQL mapping (both types and commands) is customizable via text files.

Added Values for LCG Pool:

- Standard API, already used in many Applications Frameworks and well know to Users.
- Platform-independent, multi-language API.
- Many AIDA tools ready to access/process metadata.
- Support for wide range of RDBSs.
- Global operations (chaining, merging, filtering,...) on NTuples.

Added Values for AIDA:

- SQL databases to store NTuples.

SQLTuple extends FreeHEP implementation of ITuple AIDA interface so that ITuples can be stored in an SQL database. It supports any relational DB backed via JDBC interface (configuration is provided for MySQL, PostgreSQL, McKoi and Oracle, basic tests have been performed also for Cloudscape and Hypersonic). All AIDA operations (projections, filters, evaluators,...) are supported in a standard way. Some new functions have been included on top of standard AIDA Interface.

The implementations is, in principle, ready to be used in any AIDA-complaint tool.

SQLTuple can be used to access and manage LCG Pool Event Metadata SQL storage.

SQLTuple depends on SQL only via (textual) run-time configuration files:

- **Implementation.properties** describes generic properties of SQL backends (protocol name, JDBC driver, capabilities,...).
- **Type.properties** specifies SQL-Java type mapping (for all involved SQL backends).
- **StmtSrc.properties** defines SQL commands to be used to perform AIDA functions (like ITuple.project(...), etc.).

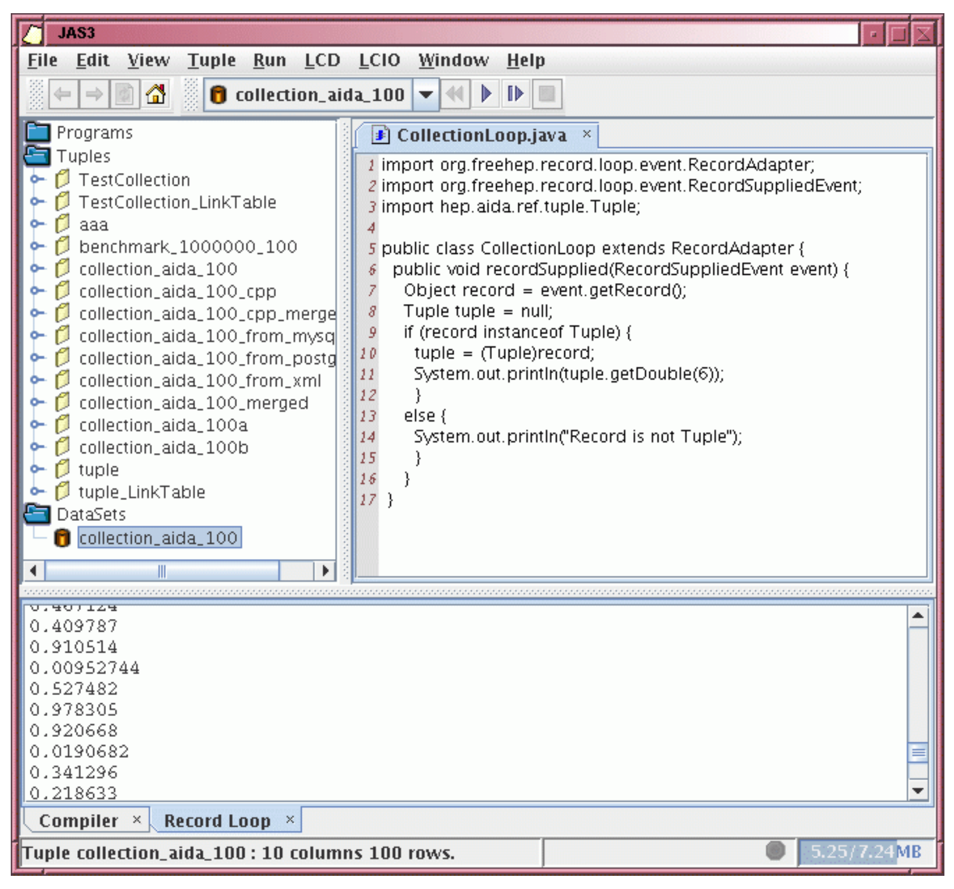
ColMan (Collection Management) provides higher-level functionality for the management of Event-level metadata (Collections).
It supports LCG Pool metadata (SQL fully, Root files in read mode). Following functions are available:

- **Filtering** – creation of subCollections based on selection string (Query).
- **Merging** – concatenation of several Collections into one.
- **Replicating** - copying Collections into different technologies and/or sites.
- **Plotting** - creation of AIDA Histograms and Clouds from Collections.
- **Selecting** - looping over Event entries selected by a Query.

ColMan functionality is exported to other languages and Frameworks via:

- ColManC – C++ proxies to ColMan classes.
- ColManWS – XML-RPC Web Service.

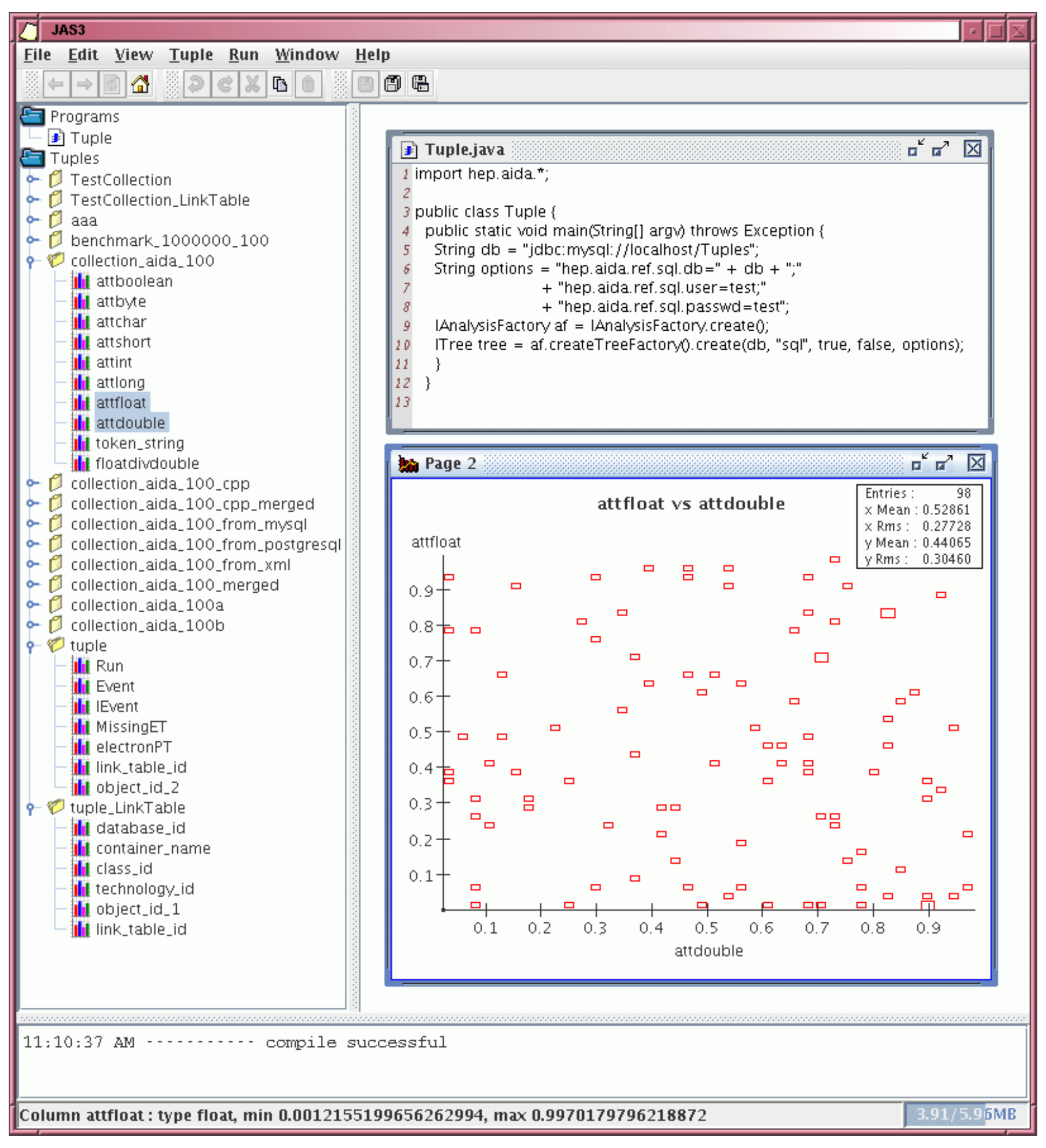
SQLTuple can be used from JAS3



```
import org.freehep.record.loop.event.RecordAdapter;
import org.freehep.record.loop.event.RecordSuppliedEvent;
import hep.aida.ref.tuple.Tuple;

public class CollectionLoop extends RecordAdapter {
  public void recordSupplied(RecordSuppliedEvent event) {
    Object record = event.getRecord();
    Tuple tuple = null;
    if (record instanceof Tuple) {
      tuple = (Tuple)record;
      System.out.println(tuple.getDouble(6));
      }
    else {
      System.out.println("Record is not Tuple");
      }
    }
 }
```

```
0.409787
0.910514
0.00952744
0.527482
0.978305
0.920668
0.0190682
0.341296
0.218633
```

Tuple collection_aida_100 : 10 columns 100 rows.



```
import hep.aida.*;

public class Tuple {
  public static void main(String[] argv) throws Exception {
    String db = "jdbc:mysql://localhost/Tuples";
    String options = "hep.aida.ref.sql.db=" + db + ";"
                   + "hep.aida.ref.sql.user=test;"
                   + "hep.aida.ref.sql.passwd=test";
    IAnalysisFactory af = IAnalysisFactory.create();
    ITree tree = af.createTreeFactory().create(db, "sql", true, false, options);
    }
  }
```

Entries :	98
x Mean :	0.52861
x Rms :	0.27728
y Mean :	0.44065
y Rms :	0.30460

```
11:10:37 AM ----------- compile successful
```

Column attfloat : type float, min 0.0012155199656262994, max 0.9970179796218872