# Programmers guide

for

# TOTem Offline DAtabase Monitor

**Authors:**
Maciej Zalewski (maciej.zalewski.mz@gmail.com)
Przemysław Dadel (pdadel@gmail.com)

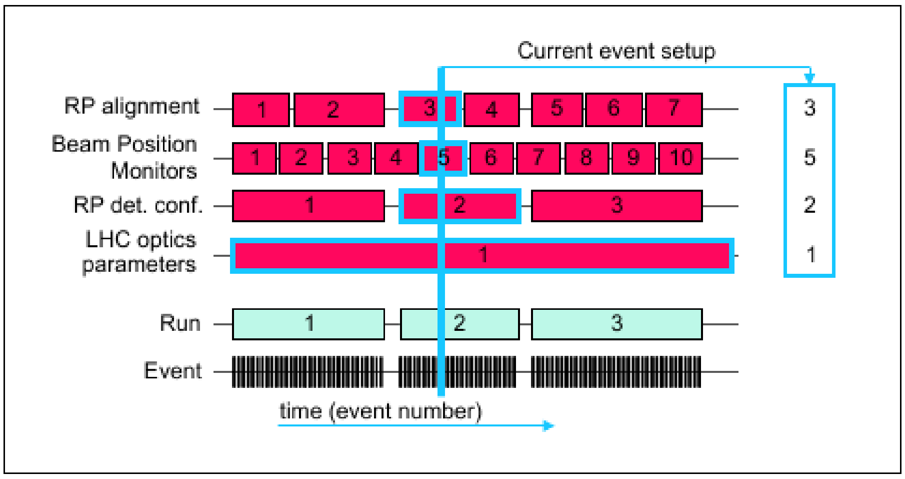
# 1. General concepts

The goal of this documentation is to provide basic information that will enable User to modify and extend the **Tot**em **O**ffline **Da**tabase **M**onitor (TOTODAM) application.

The TOTODAM application's goal is to provide graphical interface for a User to browse the content of the offline database from the point of view of Totem Offline Software. This chapter will briefly explain basic ideas behind the Totem Offline Software accessing the database. The application is not intended for changing the data itself – only for previewing the data and modifying their validity Intervals.

## *1.1. Interval of validity*

Totem Offline Software is reconstructing the experiment on the basis of events. In order to reduce the amount data acquisition in the software, there is a mechanism that implements this functionality. It uses an entity called **Interval of validity**, which represents a range of events when the kept data is valid. The Totem Offline Software retrieves data on the basis of the Interval of validity, therefore the database's central point is the mechanism that binds any data with proper Intervals of validity.



**Picture 1: The concept of Interval of validity**[1]

## *1.2. Detector structure*

Totem Offline Database stores the structure of the detector that is generic enough to store any kind of detector structure. They are divided into 2 levels:

- Low level (called Sensor Part) – the lowest level of any detector, defined by parent Structure Element and up to 3 coordinates
- Other levels (called Structure Element) – all the other levels of structure, defined by a type and string identifier, unique within a type.

For instance, in Roman Pot, the structure is kept in the following way (from the highest level):

- Arm: Structure Element (type: RP_ARM, id: 0/1)

- Station: Structure Element (type: RP_STATION, id: 00/02/10/12)
- Unit: Structure Element (type: RP_UNIT, id: 000 - 125)
- Plane: Structure Element (type: RP_SILICON_DETECTOR, id: 0000-1259)
- VFat: Structure Element (type: RP_VFAT, id: 00000-12593)
- Strip: Sensor Part (type: RP_STRIP, parent: RP_VFAT no xxxxx, coordinates (0-255)

Measurements may be combined with any Structure Element or Sensor Part.

## *1.3. Types of data*

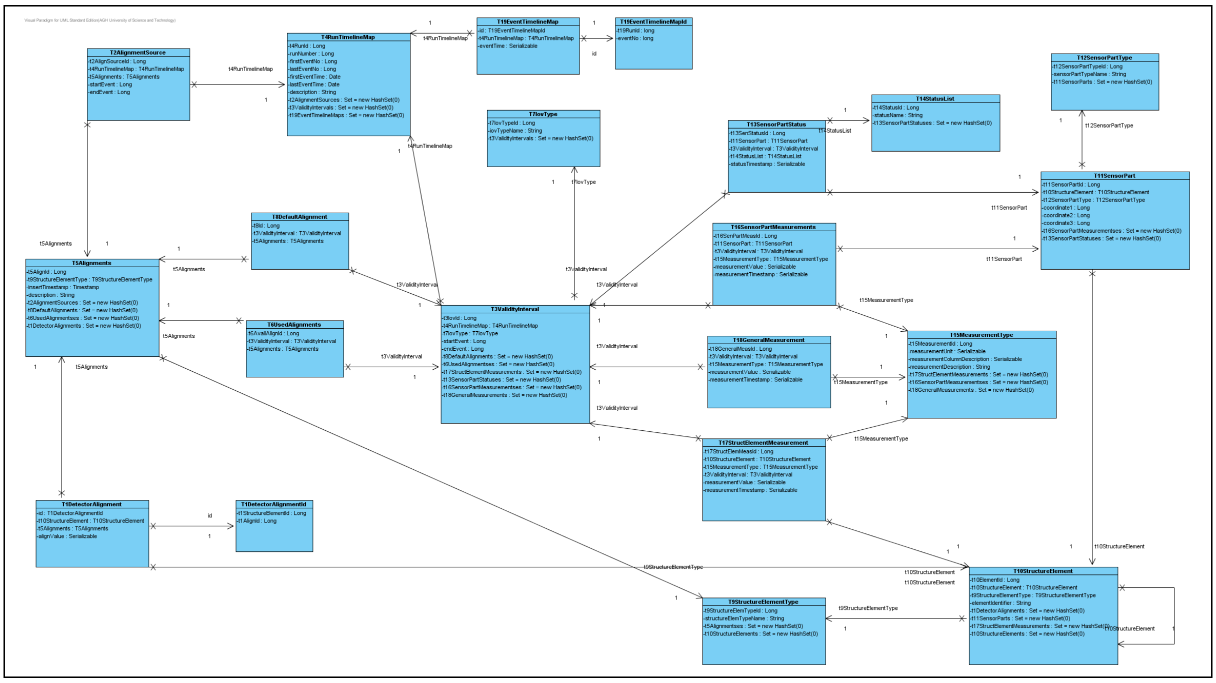
The Totem Offline Database is designed to keep any kind of data that describes the environment during the reconstructed runs. There are multiple possible sources of data. The content of the database may be divided into the following groups:

- Group data
  This kind of data is intended for storing numeric data for a group of Structure Elements. This data is managed together, has the same Interval of validity and is stored and retrieved as a whole.
- General Measurement
  This kind of data is designed for storing numeric data about global conditions or measurements for a detector as a whole.
- Structure Element Measurement
  This kind of data is designed for storing numeric data of a chosen type for each of the Structure Elements.
- Sensor Part Measurement
  This kind of data is designed for storing numeric data of a chosen type for each of the Sensor Parts.

# 2. Database overview

Before describing the application itself, it is crucial to describe the database content.
The application uses Hibernate tool to prepare a relational-object mapping. Therefore, instead of presenting the database schema, the database content will be presented as a class diagram. The classes are automatically generated, their names correspond to names of tables in the database schema, and the names of fields correspond to the names of columns in the database.

For the sake of clarity of the diagram, only class fields are presented. Each field has a getter and setter, and each class has both a non-parametric and an all-parametric constructor.

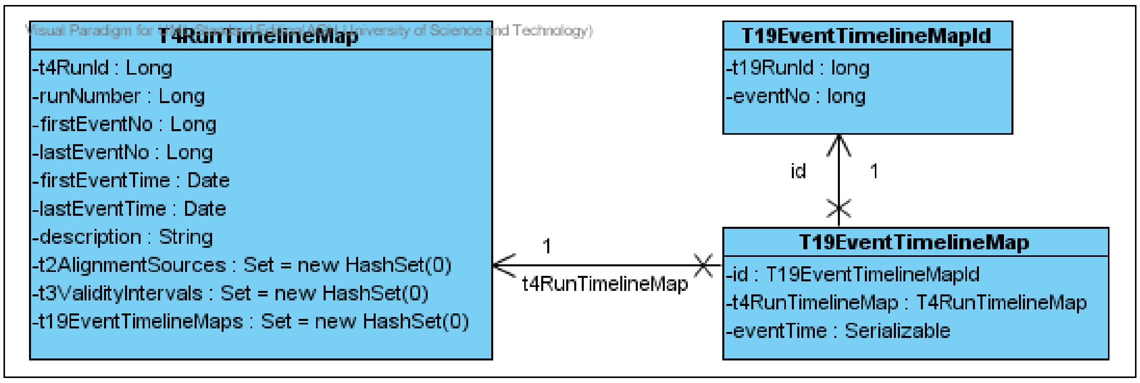


**Picture 2: Database object model**

Below, there is a full description of each class (table), with explanation for each of the fields.

## 2.1. *Time mapping objects*

There are two classes responsible for keeping record of runs and events. They also make it possible to map the time of measurement to the proper (run, event) pair, and – by that – to the Interval of validity.


Visual Paradigm for UML Standard Edition(AGH University of Science and Technology)
T4RunTimelineMap
-t4RunId : Long
-runNumber : Long
-firstEventNo : Long
-lastEventNo : Long
-firstEventTime : Date
-lastEventTime : Date
-description : String
-t2AlignmentSources : Set = new HashSet(0)
-t3ValidityIntervals : Set = new HashSet(0)
-t19EventTimelineMaps : Set = new HashSet(0)
T19EventTimelineMapId
-t19RunId : long
-eventNo : long
id
1
1
t4RunTimelineMap
T19EventTimelineMap
-id : T19EventTimelineMapId
-t4RunTimelineMap : T4RunTimelineMap
-eventTime : Serializable

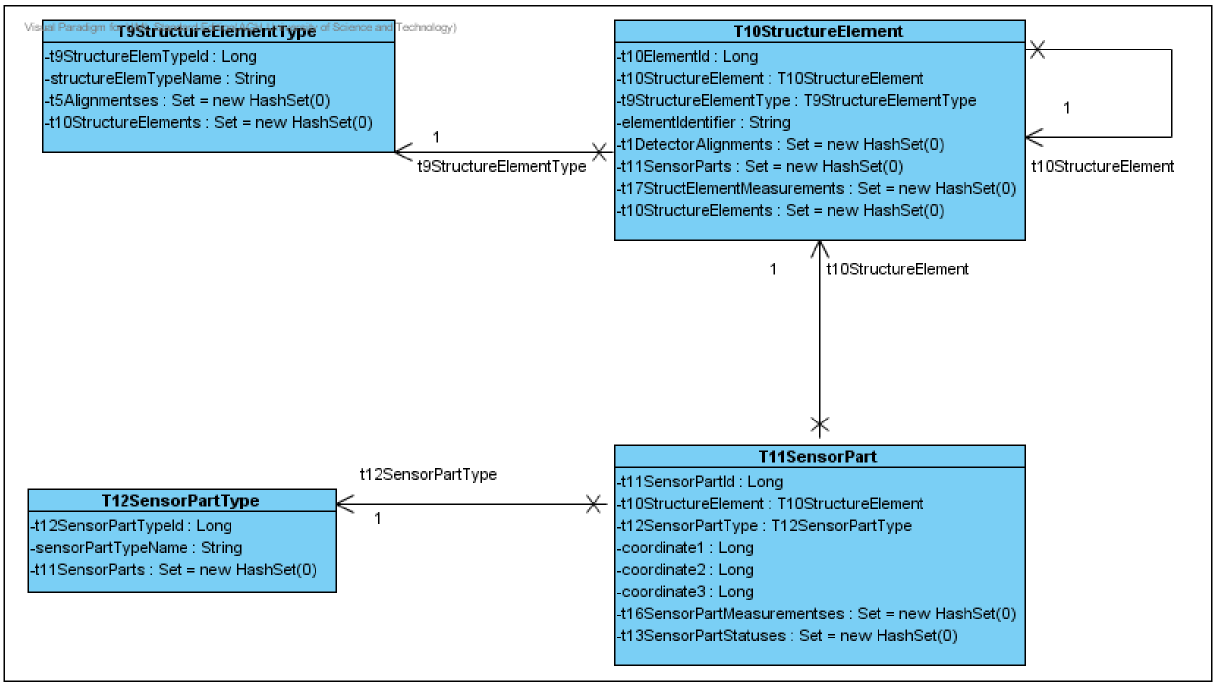
**Picture 3: Time mapping objects**

*Class name*: **T4RunTimelineMap**
The goal of this class is to keep record of registered runs.
*Fields*:
**t4RunId**: primary key of the table, number, has no domain meaning
**runNumber**: number of the run, assigned by the data sources, unique.
**firstEventNo**: number of the first event in run
**firstEventTime**: timestamp of the first event in run
**lastEventNo**: number of the last event in run
**lastEventTime**: timestamp of the last event in run
**description**: text description of the inserted run
*Outside relations*:
With T2AlignmentSources: many to one, described there
With T3ValidityIntervals: many to one, described there
With T19EventTimelineMap: many to one, described there

*Class name*: **T19EventTimelineMap**
The goal of this class is to keep mapping between timestamps and (run, event) pairs
*Fields*:
**id**: primary key of the table, of a class T19EventTimelineMapId, which consists of:
  **t19RunId**: id of a run that the given event belongs to
  **eventNo**: number of the event
**t4RunTimelineMap**: foreign key from T4 table, defining the run to which the event belongs
**eventTime**: timestamp of the event

## 2.2. Detector structure mapping objects

The database keeps information about the structure of the experiment. It is a generic, two-level structure, where the upper level (Structure element) may be recursive (so the parent of Structure element entity may be also a Structure element entity). In this part, there are also types of the elements of the experiment (eg. RP_UNIT or T2_PLANE).


T9StructureElementType
-t9StructureElemTypeId : Long
-structureElemTypeName : String
-t5Alignmentses : Set = new HashSet(0)
-t10StructureElements : Set = new HashSet(0)
1
t9StructureElementType
T10StructureElement
-t10ElementId : Long
-t10StructureElement : T10StructureElement
-t9StructureElementType : T9StructureElementType
-elementIdentifier : String
-t1DetectorAlignments : Set = new HashSet(0)
-t11SensorParts : Set = new HashSet(0)
-t17StructElementMeasurements : Set = new HashSet(0)
-t10StructureElements : Set = new HashSet(0)
1
t10StructureElement
1
t10StructureElement
t12SensorPartType
1
T12SensorPartType
-t12SensorPartTypeId : Long
-sensorPartTypeName : String
-t11SensorParts : Set = new HashSet(0)
T11SensorPart
-t11SensorPartId : Long
-t10StructureElement : T10StructureElement
-t12SensorPartType : T12SensorPartType
-coordinate1 : Long
-coordinate2 : Long
-coordinate3 : Long
-t16SensorPartMeasurementses : Set = new HashSet(0)
-t13SensorPartStatuses : Set = new HashSet(0)

**Picture 4: Detector structure mapping objects**

*Class name*: **T9StructureElementType**
This class keeps the names of structure elements
*Fields*:
**t9StructureElementTypeId**: primary key of the table, number, has no domain meaning
**structureElementTypeName**: name of the structure element, eg. RP_HYBRID
*Outside relations*:
With T5Alignments: many to one, described there
With T10StructureElement: many to one, described there

*Class name*: **T10StructureElement**
This class keeps the structure elements, eg. 4 RP_STATION's
*Fields*:
**t10ElementId**: primary key of the table, number, has no domain meaning
**t9StructureElementType**: foreign key, type of this Structure element
**t10StructureElement**: reference to the parent element, it may be null for highest-level elements
**elementIdentifier**: identifier (label) of this element that is used by the software reconstruction, eg. name of the magnet or human-readable number of RP_HYBRID
*Outside relations*:
With T1DetectorAlignment: many to one, described there
With T10StructureElement: many to one, parent-children relation, kept by t10StructureElement field
With T11SensorPart: many to one, described there
With T17StructureElementMeasurement: many to one, described there

*Class name*: **T12SensorPartType**
This class keeps the names of sensor parts (lowest-level elements of each detector)
*Fields*:
**t12SensorPartTypeId**: primary key of the table, number, has no domain meaning

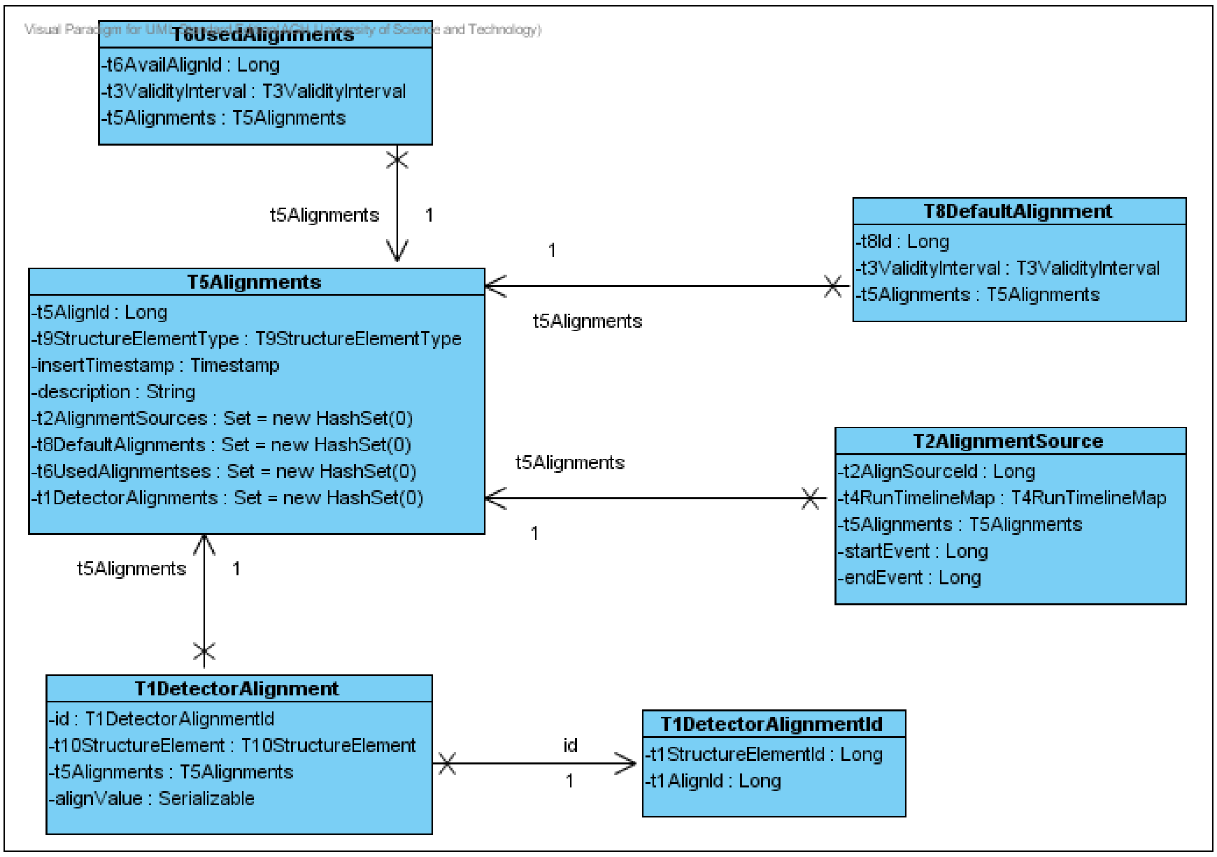
**sensorPartTypeName**: name of the sensor part, eg. RP_STRIP
*Outside relations*:
With T11SensorPart: many to one, described there

*Class name*: **T11SensorPart**
This class keeps each of the sensor parts (lowest-level elements of each detector)
*Fields*:
**t11SensorPartId**: primary key of the table, number, has no domain meaning
**t12SensorPartType**: foreign key, type of this Sensor part
**t10StructureElement**: foreign key, reference to the parent element
**coordinate1**: number, first coordinate of the sensor part within its parent element
**coordinate2**: number, second coordinate of the sensor part within its parent element
**coordinate3**: number, third coordinate of the sensor part within its parent element
*Outside relations*:
With T13SensorPartStatuses: many to one, described there
With T16SensorPartMeasurement: many to one, described there

## *2.3. Alignment mapping objects*

Alignment is a particular type of measurements, because it is computed together for the whole group of Structure elements, not separately, like the other measurements. Therefore, there are 2 classes used to map the alignment data itself: one of them keeps a thing called Detector alignment – which is an alignment of a single Structure element; the other is called Alignment and it gathers together all of the Detector alignments.

When inserting new alignment data, User should prepare an Alignment entity first, and then create multiple Detector alignment entities for the newly created Alignment.



Visual Paradigm for UML Standard Edition(AGH University of Science and Technology)
T6UsedAlignments
-t6AvailAlignId : Long
-t3ValidityInterval : T3ValidityInterval
-t5Alignments : T5Alignments
t5Alignments
1
T8DefaultAlignment
-t8Id : Long
-t3ValidityInterval : T3ValidityInterval
-t5Alignments : T5Alignments
1
t5Alignments
T5Alignments
-t5AlignId : Long
-t9StructureElementType : T9StructureElementType
-insertTimestamp : Timestamp
-description : String
-t2AlignmentSources : Set = new HashSet(0)
-t8DefaultAlignments : Set = new HashSet(0)
-t6UsedAlignmentses : Set = new HashSet(0)
-t1DetectorAlignments : Set = new HashSet(0)
T2AlignmentSource
-t2AlignSourceId : Long
-t4RunTimelineMap : T4RunTimelineMap
-t5Alignments : T5Alignments
-startEvent : Long
-endEvent : Long
t5Alignments
1
t5Alignments
1
T1DetectorAlignment
-id : T1DetectorAlignmentId
-t10StructureElement : T10StructureElement
-t5Alignments : T5Alignments
-alignValue : Serializable
id
1
T1DetectorAlignmentId
-t1StructureElementId : Long
-t1AlignId : Long

**Picture 5: Alignment mapping objects**

*Class name*: **T5Alignments**
This class keeps the Alignment entities that gather Detector alignments together
*Fields*:
**t5AlignId**: primary key of the table, number, has no domain meaning
**t9StructureElementType**: foreign key, type of Structure element. Detector alignments gathered in this Alignment should refer to Structure elements of this type
**insertTimestamp**: timestamp of creation of this alignment. Assigned automatically
**description**: text description of this Alignment
*Outside relations*:
With T1DetectorAlignment: many to one, described there
With T2AlignmentSource: many to one, described there
With T6UsedAlignments: many to one, described there
With T8DefaultAlignment: many to one, described there

*Class name*: **T1DetectorAlignment**
This class is used to keep alignment of a single detector
*Fields*:
**id**: primary key of the table of a class T1DetectorAlignmentId, consists of:
  **t1StructureElementId**: which determines which detector (Structure element) this alignment applies to
  **t1AlignId**: which determines which of the Alignments this Detector Alignment belongs to
**t10StructureElement**: foreign key, reference to the element that is aligned
**t5Alignments**: foreign key, reference to the alignment that this Detector Alignment belongs to

**alignValue**: the value of the alignment, stored in the database as a vector of doubles. The method to retrieve data from this object will be described further in the documentation

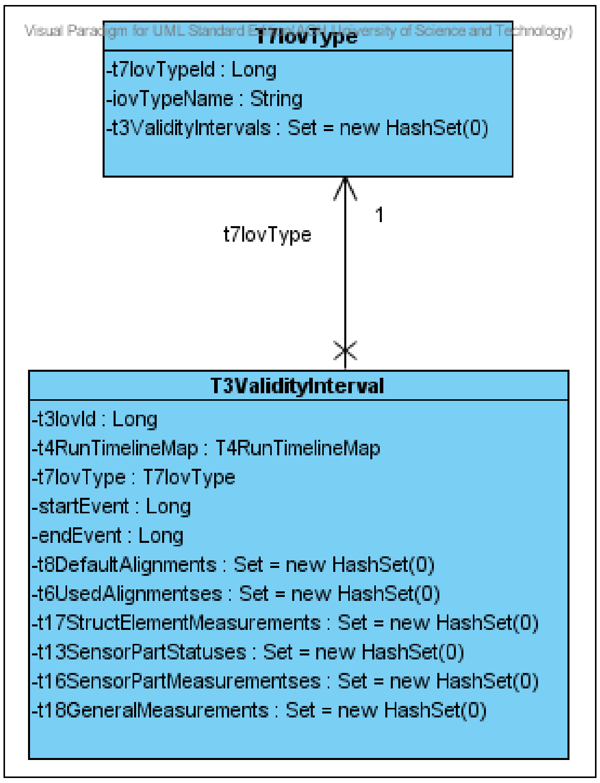
*Class name*: **T2AlignmentSource**
This class keeps information about source runs and events, on which basis the Alignment was generated
*Fields*:
**t2AlignSourceId**: primary key of the table, number, has no domain meaning
**t4RunTimelineMap**: foreign key, indicates the run that was a source for the Alignment
**t5Alignments**: foreign key, alignment for which the source is described
**startEvent**: number of the first event in the range of source events for the alignment in the given run
**endEvent**: number of the last event in the range of source events for the alignment in the given run

*Class name*: **T8DefaultAlignment**
This class binds an Interval of validity with a particular Alignment. For each of the Alignments there may be only one default alignment entry.
*Fields*:
**t8Id**: primary key of the table, number, has no domain meaning
**t3ValidityInterval**: foreign key, indicates the Interval that the Alignment is default for
**t5Alignments**: foreign key, the Alignment entity chosen for the given Interval

*Class name*: **T6UsedAlignments**
This class is a history of bindings between Intervals and Alignment. Entries for the interval present in this table keep Alignments which are preferable choice.
*Fields*:
**t6AvailAlignId**: primary key of the table, number, has no domain meaning
**t3ValidityInterval**: foreign key, indicates the Interval that the Alignment was used for
**t5Alignments**: foreign key, the Alignment entity usable for the given Interval

## 2.4. Validity interval mapping objects

Interval of validity is the central point of the offline software and the database. The Interval information is stored in one table and uses one dictionary table that defines the type of Interval. The type of Interval is a crucial concept in the TOTODAM application, as it is the basis for the way the data is displayed.


Visual Paradigm for UML Standard Edition(Vilnius University of Science and Technology)
T7IovType
-t7IovTypeId : Long
-iovTypeName : String
-t3ValidityIntervals : Set = new HashSet(0)
t7IovType
1
T3ValidityInterval
-t3IovId : Long
-t4RunTimelineMap : T4RunTimelineMap
-t7IovType : T7IovType
-startEvent : Long
-endEvent : Long
-t8DefaultAlignments : Set = new HashSet(0)
-t6UsedAlignmentses : Set = new HashSet(0)
-t17StructElementMeasurements : Set = new HashSet(0)
-t13SensorPartStatuses : Set = new HashSet(0)
-t16SensorPartMeasurementses : Set = new HashSet(0)
-t18GeneralMeasurements : Set = new HashSet(0)

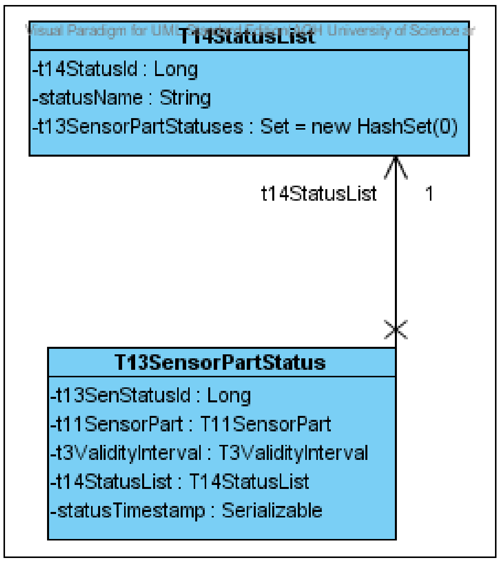
**Picture 6: Validity interval mapping objects**

*Class name*: **T7IovType**
This class keeps information about the types of Interval of validity
*Fields*:
**t7IovTypeId**: primary key of the table, number, has no domain meaning
**iovTypeName**: name of the Interval type, label
*Outside relations*:
With T3ValidityInterval: many to one, described there

*Class name*: **T3ValidityInterval**
This class keeps information about Intervals of validity
*Fields*:
**t3IovId**: primary key of the table, number, has no domain meaning
**t4RunTimelineMap**: foreign key, determines the run this Interval belongs to
**t7IovType**: foreign key, determines the type of this Interval
**startEvent**: number of the first event of this Interval
**endEvent**: number of the last event of this Interval
*Outside relations*:
With T6UsedAlignments: many to one, described there
With T8DefaultAlignment: one to one, described there
With T13SensorPartStatus: many to one, described there
With T16SensorPartMeasurement: many to one, described there
With T17StructureElementMeasurement: many to one, described there
With T18GeneralMeasurement: many to one, described there

## 2.5. *Sensor part status mapping objects*

Sensor part status is a specific measurement, as it is the only data that may be modified from TOTODAM application. The mapping objects for this functionality are very simple, one of them is a list of possible statuses, and the other connects the Interval with a particular sensor part and its status.


T14StatusList
-t14StatusId : Long
-statusName : String
-t13SensorPartStatuses : Set = new HashSet(0)
t14StatusList
1
T13SensorPartStatus
-t13SenStatusId : Long
-t11SensorPart : T11SensorPart
-t3ValidityInterval : T3ValidityInterval
-t14StatusList : T14StatusList
-statusTimestamp : Serializable

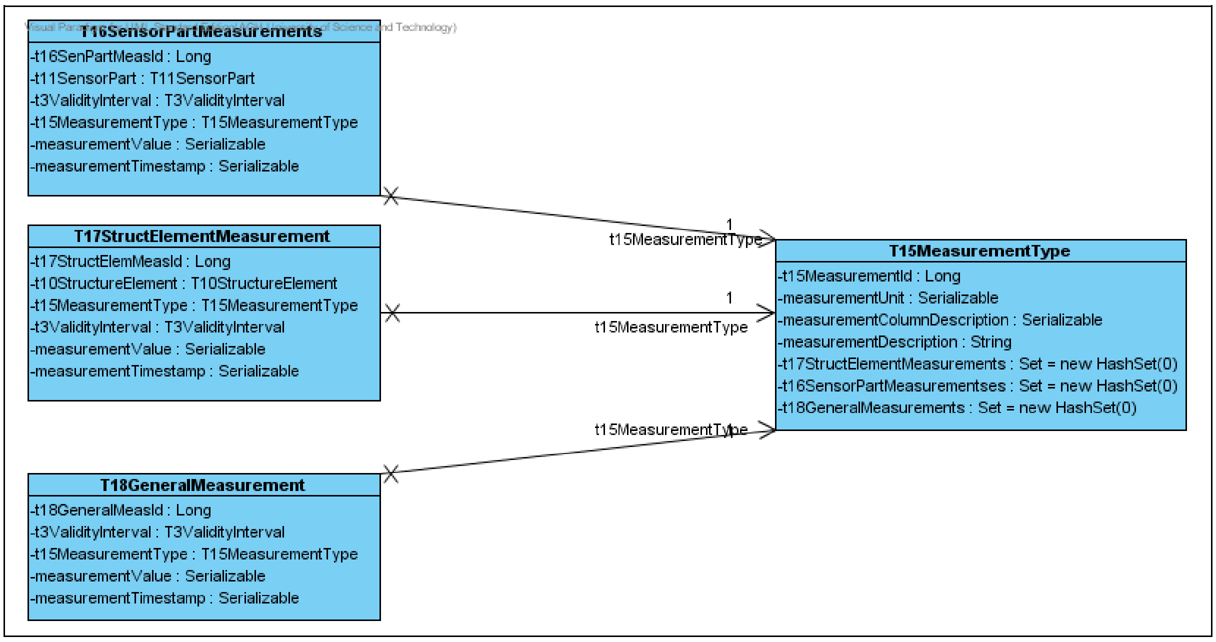
**Picture 7: Sensor part status mapping objects**

*Class name*: **T14StatusList**
This class keeps the name of possible statuses
*Fields*:
**t14StatusId**: primary key of the table, number, has no domain meaning
**statusName**: a label that describes the status, eg. NOISY
*Outside relations*:
With T13SensorPartStatus: many to one, described there

*Class name*: **T13SensorPartStatus**
This class keeps the information what is the status of each Sensor part in the given Interval
*Fields*:
**t13SenStatusId**: primary key of the table, number, has no domain meaning
**t11SensorPart**: foreign key of the Sensor parts table, informs which of the Sensor parts the status applies to
**t3ValidityInterval**: foreign key of the Validity interval table, binds the given entity with the Interval
**t14StatusList**: foreign key of the Status list table, specifies the status of the Sensor part
**statusTimestamp**: the timestamp to which the status apply to

## 2.6. *Measurement mapping objects*

Any measurements different than those mentioned above are kept in one of three tables (mapped to the classes), depending on the element of the structure that this measurement is connected with. There are three types of measurements: general (not related to any structure element), related to a Structure element and related to a Sensor part.


T16SensorPartMeasurements
-t16SenPartMeasId : Long
-t11SensorPart : T11SensorPart
-t3ValidityInterval : T3ValidityInterval
-t15MeasurementType : T15MeasurementType
-measurementValue : Serializable
-measurementTimestamp : Serializable
T17StructElementMeasurement
-t17StructElemMeasId : Long
-t10StructureElement : T10StructureElement
-t15MeasurementType : T15MeasurementType
-t3ValidityInterval : T3ValidityInterval
-measurementValue : Serializable
-measurementTimestamp : Serializable
T18GeneralMeasurement
-t18GeneralMeasId : Long
-t3ValidityInterval : T3ValidityInterval
-t15MeasurementType : T15MeasurementType
-measurementValue : Serializable
-measurementTimestamp : Serializable
1
t15MeasurementType
1
t15MeasurementType
t15MeasurementType
T15MeasurementType
-t15MeasurementId : Long
-measurementUnit : Serializable
-measurementColumnDescription : Serializable
-measurementDescription : String
-t17StructElementMeasurements : Set = new HashSet(0)
-t16SensorPartMeasurementses : Set = new HashSet(0)
-t18GeneralMeasurements : Set = new HashSet(0)

**Picture 8: Measurement mapping objects**

*Class name*: **T15MeasurementType**
This class keeps available measurements type name. As each of the measurements is a vector of numbers (double), it also contains list of labels for each of the vector elements, and also units for each of the values
*Fields*:
**t15MeasurementId**: primary key of the table, number, has no domain meaning
**measurementUnit**: array of strings, containing a list of units for each of the values
**measurementColumnDescription**: array of strings, containing a list of labels for each of the values
**measurementDescription**: a label for this measurement type
*Outside relations*:
With T16SensorPartMeasurement: many to one, described there
With T17StructureElementMeasurement: many to one, described there
With T18GeneralMeasurement: many to one, described there

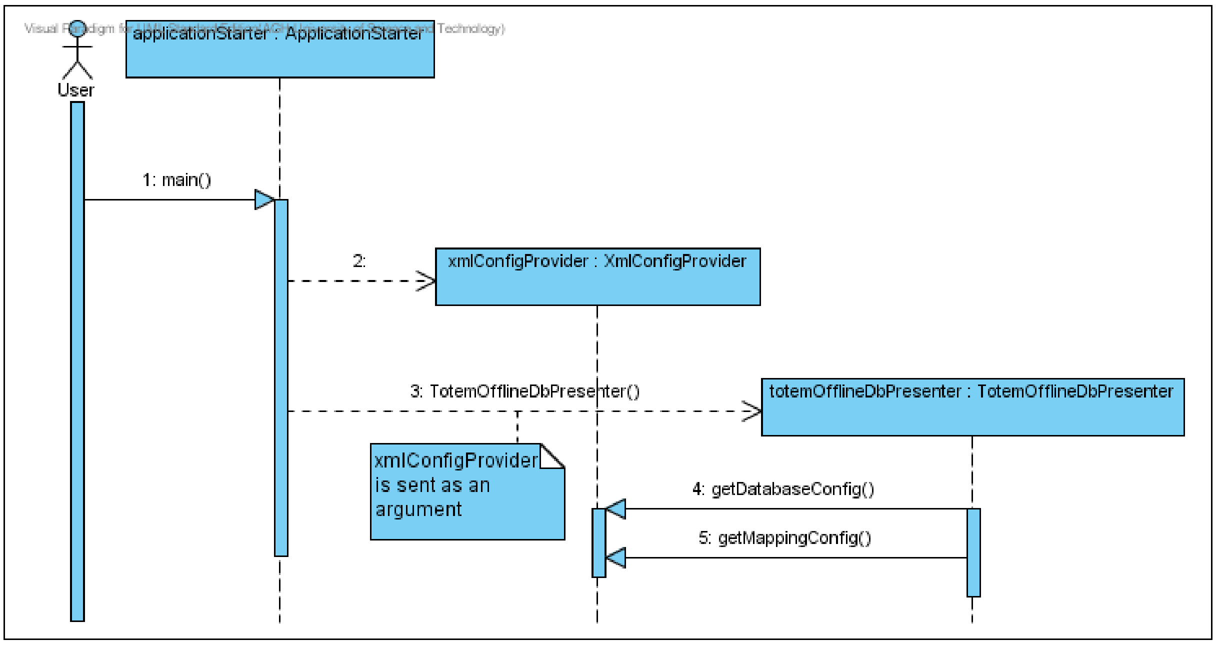
*Class name*: **T16SensorPartMeasurement**
This class keeps single measurement (vector of doubles) connected with the Sensor part
*Fields*:
**t16SenPartMeasId**: primary key of the table, number, has no domain meaning
**t11SensorPart**: foreign key, defines which of the Sensor parts this measurement applies to
**t3ValidityInterval**: foreign key of the Validity interval table, binds the given measurement with its validity interval
**t15MeasurementType**: foreign key for the measurement type table, defines the kind of measurement
**measurementValue**: vector of doubles that keeps the measurement itself
**measurementTimestamp**: timestamp of the measurement. Attention: it may not be the timestamp of inserting to the database!

*Class name*: **T17StructElementMeasurement**
This class keeps a single measurement (vector of doubles) connected with the Structure element
*Fields*:
**t17StructElemMeasId**: primary key of the table, number, has no domain meaning
**t10StructureElement**: foreign key, defines which of the Structure elements this measurement applies to
**t3ValidityInterval**: foreign key of the Validity interval table, binds the given measurement with its validity interval
**t15MeasurementType**: foreign key for the measurement type table, defines the kind of measurement
**measurementValue**: vector of doubles that keeps the measurement itself
**measurementTimestamp**: timestamp of the measurement. Attention: it may not be the timestamp of inserting to the database!

*Class name*: **T18GeneralMeasurement**
This class keeps single measurement (vector of doubles) that is general – not related to any element of the detector structure
*Fields*:
**t18GeneralMeasId**: primary key of the table, number, has no domain meaning
**t3ValidityInterval**: foreign key of the Validity interval table, binds the given measurement with its validity interval
**t15MeasurementType**: foreign key for the measurement type table, defines the kind of measurement
**measurementValue**: vector of doubles that keeps the measurement itself
**measurementTimestamp**: timestamp of the measurement. Attention: it may not be the timestamp of inserting to the database!

# 3. TOTODAM application skeleton

The TOTODAM application entry point is in class ***ch.cern.totem.offline.ApplicationStarter***. It prepares the configuration reader (that reads configuration from the XML file, may be substituted) and starts the *Main Presenter* (***TotemOfflineDbPresenter***).


Visual Paradigm for UML Standard Edition(CERN (European Organization for Nuclear Research) Technology)
User
applicationStarter : ApplicationStarter
1: main()
2:
xmlConfigProvider : XmlConfigProvider
3: TotemOfflineDbPresenter()
totemOfflineDbPresenter : TotemOfflineDbPresenter
xmlConfigProvider is sent as an argument
4: getDatabaseConfig()
5: getMappingConfig()

**Picture 9: TOTODAM startup**

The *Main Presenter* creates a database connector that enables the communication with the db on the basis of the configuration read from an XML file. It also provides the functionality of presenting dialog boxes for creating new:

- Interval types
- Measurement types
- Run entry

and

- Modifying database settings

Dialogs for this can be found in package ***ch.cern.totem.offline.dbmonitor.view***.

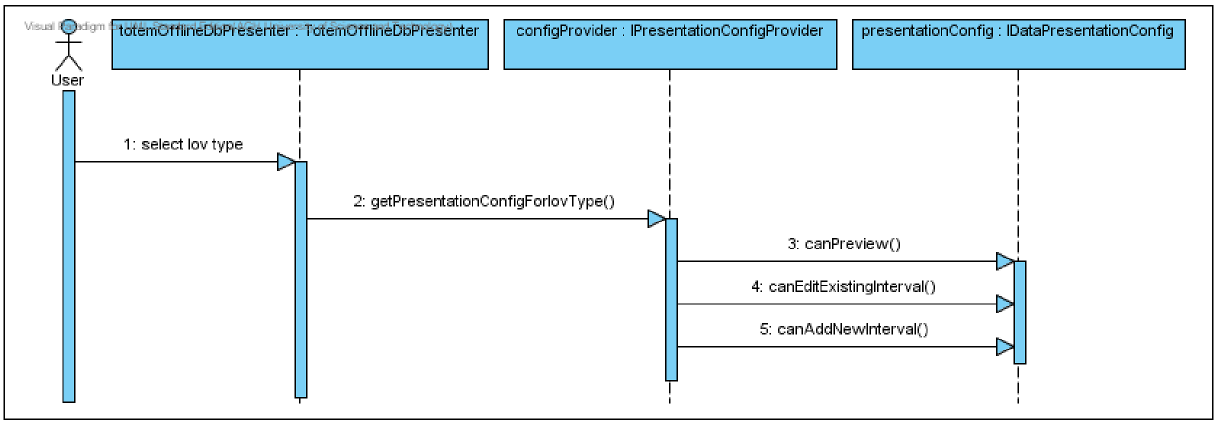
The database connector may be found in package ***ch.cern.totem.offline.dbmonitor.database***. There is a sub-package: ***wrapper*** which contains wrappers for the persistence items. Those wrappers are used in swing classes for displaying persistence items – their main functionality is to have the ***toString*** method overridden, so that they present better. It was not done within the persistence classes, as they are generated automatically and any changes within them may be overridden.

## 4. Data presentation

The main functionality of the TOTODAM application is to present data stored in the database. However, it is not possible to foresee every kind of data that will be kept, and presenting them in the same way may not be the best method. Therefore, the basic assumption was to make it easy to add new way of presenting.

The crucial part of presentation configuration is the type of validity interval. In the application configuration, there is a mapping between the Interval type name and the class that provides the presentation for the given type. When the user selects a type of Interval, the *Main*

*Presenter* refers to its data presentation configuration provider to receive proper data presentation configuration.


User
totemOfflineDbPresenter : TotemOfflineDbPresenter
configProvider : IPresentationConfigProvider
presentationConfig : IDataPresentationConfig
1: select lov type
2: getPresentationConfigForlovType()
3: canPreview()
4: canEditExistingInterval()
5: canAddNewInterval()

**Picture 10: Presentation choice**

## 4.1. Data presentation configuration provider

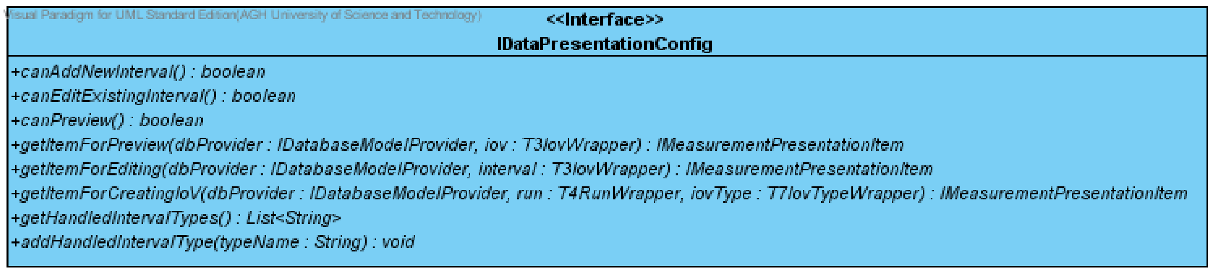
The data presentation configuration provider is the class that provides *Main Presenter* with appropriate presentation configurations for particular Interval type.

There is a skeleton implementation for this configuration provider, in the form of an abstract class which can be found in:
***ch.cern.totem.offline.dbmonitor.config.PresentationConfigProviderSkeleton***
It is an example of usage of the Template Method design pattern. The skeleton implements the method that returns the configuration provider on the basis of its protected field, but the field itself is populated by an abstract method implemented by a sub-class. It can be dynamically changed during the execution of the program, what leaves this element of software open for further extending.

There also is a single implementation of a class that extends the skeleton above – it is configured from the mapping file. The implementation resides in the same package and its class name is ***ConfigurablePresentationConfigProvider***. It is instantiated from the *Main Presenter* with the mapping provided from the file. Further details on the mapping configuration may be found in the User's Guide.

## 4.2. Data presentation configuration

The data presentation configuration provides *Main Presenter* with the possibility of determining what actions are available for particular types of the Interval. It also is used to provide *Main Presenter* with the presentation item, used to present the particular data. The most obvious way of creating such presentation configuration is to provide one for each of the interval type. This is the first item that needs to be implemented when adding new Interval type presentation to the application.



```
<<Interface>>
IDataPresentationConfig
+canAddNewInterval() : boolean
+canEditExistingInterval() : boolean
+canPreview() : boolean
+getItemForPreview(dbProvider : IDatabaseModelProvider, iov : T3IovWrapper) : IMeasurementPresentationItem
+getItemForEditing(dbProvider : IDatabaseModelProvider, interval : T3IovWrapper) : IMeasurementPresentationItem
+getItemForCreatingIoV(dbProvider : IDatabaseModelProvider, run : T4RunWrapper, iovType : T7IovTypeWrapper) : IMeasurementPresentationItem
+getHandledIntervalTypes() : List<String>
+addHandledIntervalType(typeName : String) : void
```

**Picture 11: Data presentation configuration interface**

The example implementations can be found in the package ***ch.cern.totem.offline.dbmonitor.config.dataconfig***. They are very compact and should be easy to implement. Their name should then appear in the XML file for application initialisation, along with the proper Interval type they correspond to.

In case there is no configuration for some type of the Interval, the skeleton of configuration provider returns a presentation configuration item called ***UnsupportedDataPresentationConfig*** that disables all actions for the Interval type.

## 4.3. Presentation item

The presentation item is the element that is responsible for providing display of a given Interval. However, due to possible complications, especially with the database management, there is a default implementation that is suggested to be used for creating new presentation items.

### 4.3.1. Dialog presentation item

The default implementation of the presentation item resides in package ***ch.cern.totem.offline.dbmonitor.view.datadisplay.DefaultMeasurementPresentationDialog***. The dialog class specifies an interface ***IMeasurementPresentationPanel*** which must be implemented by any class that is to be within the dialog. This interface specifies the following methods:

1. public JPanel getPanelToEmbed();
2. public boolean validatePanelContent();
3. public List<Object> getObjectsToPersist();
4. public String getTitle();

The first method returns the ***JPanel*** object that will be embedded on the dialog.
The second one is probably the most important one – the implementing class should then validate whether the modified data is valid (this method is called before making any changes made by the User using the panel persistent).
The third method is also very important – during modifications via the panel, it is possible that some new objects will be created. Those objects must be returned by this method, otherwise they will not be stored in the database.
The last method affects only the title of the dialog.

There are five implementations of ***IMeasurementPresentationPanel*** in the TOTODAM application – one for each kind of data stored in the database. The most advanced one is certainly the one with Alignments, thus it may be referred to as a good example. All of the implementations are located in the package ***ch.cern.totem.offline.dbmonitor.view.datadisplay***.

# 5. Hibernate usage

The crucial part of developing any further code in the TOTODAM application is understanding the concept of mapping the database schema onto the classes. The classes themselves can be generated by the Hibernate tool, for example the one integrated with NetBeans IDE. The other part is the mapping definition, which is also generated, however, it tends to require some improvements:

- Oracle Timestamps are mapped onto Serializable instead of java.sql.Timestamp, which causes the application to fail while retrieving any data from the database. The "type" property in proper hibernate\class\property entry in hbm.xml mapping file should be changed from *serializable* (generated by default by the tool) to *java.sql.Timestamp*
- Any custom types are mapped onto Serializable. It also causes the application to fail. The solution is described in the chapter below.

## *5.1. Retrieval of ARRAY data*

The Totem Offline Software Database uses four user-definied types for storing vectors. In TOTODAM application, there is a proper mapping of this structures onto the entities retrieved from the database. To see how it works, please refer to the code, for example ALIGNMENT_VECTOR mapping resides in:

- ch.cern.totem.offline.persistence. T1DetectorAlignment class (TotemPersistence.jar)
- ch.cern.totem.offline.persistence.datatype.OracleAlignmentVector (as above)
- TotemPersistence.jar, in file META-INF\mapping\T1DetectorAlignment.hbm.xml

The example of usage is quite complex. A good example of data retrieval can befound in **ch.cern.totem.offline.dbmonitor.view.datadisplay.GeneralMeasurementsPanel** in methods getMeasurementFromArray and getMeasurementName. Setting the data is done in a much simpler way, ascan be seen in **ch.cern.totem.offline.dbmonitor.view.MeasurementTypeCreationDialog**, in method jButtonOkActionPerformed (location of this code may change)

Further information may be found also on hibernate web page[2]

[1] From PhD thesis of dr Hubert Niewiadomski (http://cdsweb.cern.ch/record/1131825?ln=en)
[2] https://www.hibernate.org/261.html