Distrib Parallel Databases (2016) 34:65–99
DOI 10.1007/s10619-014-7165-7

SODA: A framework for spatial observation data analysis

Sebastián Villarroya · José R. R. Viqueira · Manuel A. Regueiro · José A. Taboada · José M. Cotos

Published online: 5 November 2014

Abstract Very large amounts of geospatial data are daily generated by many observation processes in different application domains. The amount of produced data is increasing due to the advances in the use of modern automatic sensing devices and also in the facilities available to promote crowdsourcing data collection initiatives. Spatial observation data includes both data of conventional entities and also samplings over multi-dimensional spaces. Existing observation data management solutions lack declarative specification of spatio-temporal analytics. On the other hand, current data management technologies miss observation data semantics and fail to integrate the management of entities and samplings in a single data modeling solution. The present paper presents the design of a framework that enables spatio-temporal declarative analysis over large warehouses of observation data. It integrates the management of entities and samplings within a simple data model based on the well known mathematical concept of function. Observation data semantics are incorporated into the model with appropriate metadata structures.

S. Villarroya · J. R. R. Viqueira (✉) · M. A. Regueiro · J. A. Taboada · J. M. Cotos
Computer Graphics and Data Engineering Group (COGRADE), Centro Singular de Investigación en Tecnoloxías da Información (CITIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain
e-mail: jrr.viqueira@usc.es

S. Villarroya
e-mail: sebastian.villarroya@usc.es

M. A. Regueiro
e-mail: manuelantonio.regueiro@usc.es

J. A. Taboada
e-mail: joseangel.taboada@usc.es

J. M. Cotos
e-mail: manel.cotos@usc.es

Keywords Spatial data · Observation data · Sensor data · Data analysis · Data warehouse

1 Introduction

Nowadays, an increasing number of automatic data acquisition devices (sensors) are observing every day more variables in every day more applications domains (home automation, industrial process monitoring, health care, environmental monitoring, etc.). In many cases, the location of each observation in some reference space is an important piece of metadata that must be used during data analysis. This is always the case in applications in the area of environmental data management. As an example, a huge amount of data is generated on a daily basis by earth observation sensors on board of different satellites. Together with social media text data, the above observation data is one of the main data sources subject for current and future application of consolidated and emerging Big Data technologies.

According to the Observations and Measurements (O&M) conceptual schema [1], properties of *Entities* are either exact values assigned by some authority (name of a river, geometry of a municipality, etc.) or estimated by some (Observation) *Process* (temperature of sea surface, weight of a person, etc.). In order to adequately interpret values of *Observed Properties*, relevant observation metadata has to be recorded. Thus, it is usually mandatory to register characteristics of the specific *Process* used to generate the value. It is also mandatory to know the time instant at which the observed value starts to be valid for the *Entity* (*phenomenonTime* in [1]). Notice for example that the analysis of water (*Process*) from a river (*Entity*) might be performed in a laboratory some days after the water was obtained (*phenomenonTime*). The above metadata provides observation semantics to such property values. Observation *Processes* of very different nature might be found in real application scenarios, including physical devices (sensors), tasks performed by people and data processing algorithms. A classification of *Processes* is proposed in [2]. Two specific characteristics of a *Process* determine the type of observation data that it produces.

- A *Time-triggered Process* is performed at some predefined time frequency and therefore the observation data that it produces has the form of a regular sampling in the temporal domain. An example of this is the sampling of air temperature obtained by the temperature sensor of a meteorological station every ten minutes. On the other hand, *Event-triggered Processes* might start at any time instant, being fired by some event. For example, temperature and viscosity of volcanic lava might be measured by a volcanologist at any moment.
- If we restrict to sensors (physical *Processes* according to [2]), in-situ sensors observe *Entities* that are located in their same spatial position and they generate a single observed value at each time instant. An example of an in-situ sensor is a temperature sensor that might be installed in a meteorological station (static platform) or in a radio-sounding device (mobile platform). On the other hand, remote sensors observe *Entities* that are far away from its location and they use to generate various observed values (one for each observed *Entity*) at each time instant. An example of a remote sensor installed in a static platform is an Acoustic

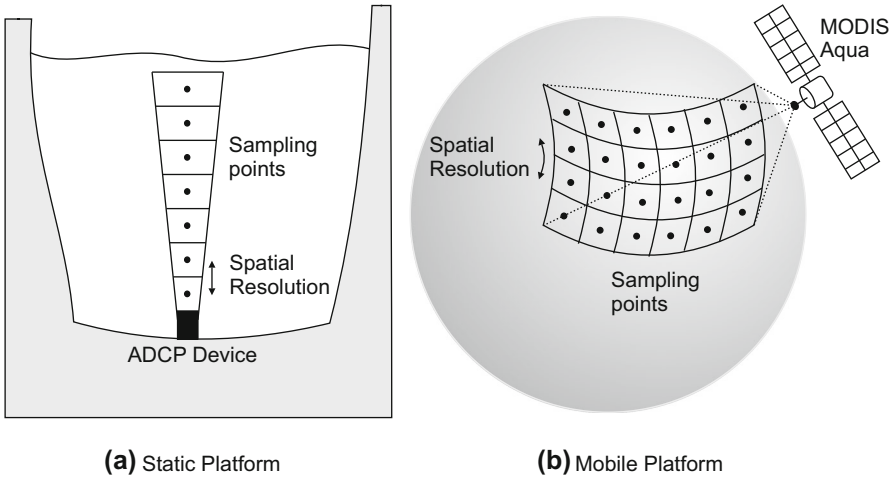

Fig. 1 Illustration of 1D and 2D spatial samplings

Doppler Current Profiler (ADCP), that produces at each time instant a 1D sampling of water current velocities along consecutive discrete locations of a straight line profile, either horizontal or vertical. Figure 1a illustrates a vertical section of a body of water in whose bottom a ADCP sensor is installed . An example of a remote sensor installed in a mobile platform is the Moderate-resolution Imaging Spectroradiometer (MODIS) sensor installed in Terra and Aqua NASA satellites (see Fig. 1b for an illustration). The produced data includes a 2D regular sampling of sea surface temperatures (with a spatial resolution of 4 km) every 8 days. 2D regular samplings are called *Rasters* in the area of geographic data management.

As a consequence of the above, to effectively manage observation data, a system must provide the following general functionalities.

1. Support for the management of conventional Entity/Relationship (ER) data related to non-observed properties of *Entities*.
2. Support for observation data semantics provided by relevant observation metadata of *Observed Properties* of *Entities*. Thus, *Entity Types* together with their relevant conventional and *Observed Properties* and *Process Types* of any type should be declared to the system with the help of some declarative observation data definition language. Observation metadata of a given observed value should be provided during data insertion, including the instance of the *Process Type* (for example a specific sensing device) used to get the value and its phenomenonTime.
3. Support for the management of sampled data over temporal, spatial (1D and 2D) and spatio-temporal domains.

To the best of these authors knowledge, none of the available technologies and approaches found in data management literature provide support for all the above functionalities. In particular, observation data semantics are only explicitly supported by standards of the Open Geospatial Consortium (OGC), Sensor Web Enablement

(SWE) initiative [1–3] and by specific observation data models and ontologies [4–6]. However, the support of declarative analysis over observation data is out of the scope of all those models and standards. Currently available Geographic Information System (GIS) tools [7] provide support for the recording and processing of conventional and spatial data, including Rasters, however, they lack support for declarative data analysis. Various systems have been developed for the declarative management of data streams of sensor data [8,9], however, sampled data is not supported in these systems. Many research approaches have been proposed in the area of spatial databases [10,11] and relevant functionality has been added to ISO SQL standard [12], which is implemented by well known DBMSs [13]. Currently, these tools provide support for declarative querying of spatial data, including limited support for 2D Rasters. Spatial extensions have also been implemented in NoSQL [14] and high performance Data Warehouse [15] tools. However, sampled data is not supported in these systems. Declarative analysis over very large collections of sampled data is supported by array data managers [16,17]. However, declarative analysis of relational data is not user friendly with array data structures. Finally, the integrated management of relational and array data is attempted in [18]. To achieve this, the user has to deal with both relational and array semantics, which in these authors opinion is not user friendly.

Based on all the above limitations of currently available systems and approaches, the objective of the present work is the design of Spatial Observation Data Analysis (SODA), a framework for declarative spatio-temporal analysis in very large warehouses of spatial observation data. The framework seamlessly integrates entity based and sampling data in a simple data model based on the well known mathematical concept of function. The model incorporates also observation data semantics. A main handicap is that it is not based on relational or object oriented paradigms, however the data model is very simple and the language combines logical and functional constructors already present in other well known languages like XQuery. The contributions of this paper may be summarized as follows.

- Formalization of a data model for the integrated management of entity and sampling data, using a new hybrid logical-functional paradigm.
- Definition of a spatio-temporal declarative data analysis language for the above data model.
- Definition of a data warehouse data model with support for observation data semantics. Application of the above language to the declarative definition of new observation *Processes* that are executed by the framework during observation data load.
- Brief discussion of physical level issues related to the column-oriented implementation of the framework that is currently being undertaken. The implementation must exploit parallelization in current multi-core hardware architectures.

The remainder of this paper is organized as follows. Section 2 discusses and evaluates technologies and research approaches related to the present one. The formalization of the data model for observation data warehouses is given in Sect. 3. Section 4 describes the spatio-temporal analysis language provided by the framework. Column-oriented implementation issues are discussed in Sect. 5. Finally, Sect. 6 concludes the paper and outlines pieces of further work.

2 Related work

Various related approaches and technologies are now described and compared with respect to the following criteria, which is derived from the generic functionalities that an observation management system must support.

1. *Direct support of observation semantics*: In order to perform effective analysis and interpretation of values of *Observed Properties* of *Entities*, some important metadata have to be recorded and linked to the observed values. In particular, at least, it is important to record a reference to the specific *Process* used to generate the observation and also the time at which the observed value applies to the *Entity*. *Process* instances have to be classified into *Process Types* as *Entities* are also classified into *Entity Types* in the classical E/R model. Besides, the system should also support the recording of properties of *Processes*. Thus, for example, in a meteorological observation domain, a *Process Type* "TemperatureSensor" could have self described properties "DeviceId" and "InstallationDate". Each value of the *Observed Property* "AirTemperature" of each Meteorological Station *Entity* should be linked to the specific instance of "TemperatureSensor" that was used to measure it.
2. *Support for the management of sampled data*: Beyond the classical E/R data, an observation data management approach must also provide data structures and operations that enable the efficient processing of sampled data. As it was already reported in the introduction, temporal samplings are generated by *Time-triggered Processes*, whereas spatial samplings are usually produced by remote sensors. It is noticed that the use of classical relational-based models for sampled data results in either highly inefficient approaches or complex nested models as will be shown below in this section.
3. *Support for multi-resolution temporal and spatial data*: The observation data generated by currently available sensors is produced with different temporal and spatial resolutions. Thus for example, MODIS generates sea surface temperature data with a temporal resolution of 8 days and with a spatial resolution of 4 km. An observation data management approach should provide a data type system that simplifies the transformation between different temporal and spatial resolutions during the evaluation of operations.
4. *Simple data modeling approach*: In the context of this evaluation, a simple data model is the one that uses just one non-nested data structure. Thus, for example, a relational model is considered simple, whereas an object-relational one is not, as nested arrays and collections are supported. Besides, data models that use different data structures with different semantics for different types of data are also considered non-simple. It is obvious that the efficient implementation of a nested data model is far more complicated than the implementation of a non-nested one. On the other hand, it is also clear that having to deal with various data structures with different semantics leads to interfaces that are not user friendly.
5. *Model based on a well known paradigm*: The definition of data models that are based on well known paradigms as the relational one allows to take advantage of many years of user experience, improving their learning curve.

6. *Stream Processing approach*: If there are real-time requirements and the amount of data to be recorded is not large then the approach must integrate the efficient processing of input data streams with small stored data structures to produce output data streams. Stream processing approaches are commonly known as Complex Event Processing (CEP) (Information Flow Processing Systems in [19]) and they rely on the evaluation of Continuous Query Language (CQL) expressions [20,21].
7. *On Line Transaction Processing (OLTP) approach*: If real-time requirements are present with simpler temporal patterns but large amounts of data have to be recorded, then an OLTP processing approach is required. This approach is traditionally supported by conventional DBMSs for reasonably large data collections and provided by both NoSQL [14,22] and NewSQL [23] solutions in the new era of Big Data Management.
8. *On Line Analytical Processing (OLAP) approach*: Non-real time analytics over very large data sets is supported by OLAP approaches provided by Business Intelligence solutions over Data Warehouse technologies. High performance implementations include Hewlett-Packard Vertica [24], which is an evolution of CStore [25] and the open source Monetdb database [15]. The efficient implementations of these Big Data solutions are based on recent research on column-oriented technologies. The key feature of these approaches is that relational data is recorded by columns, instead of the classical row storage. This enables on the one hand the application of efficient compression techniques to the data and even to perform some processing over compressed data and on the other hand avoids retrieving from storage columns that are not involved in computations. The main drawback is that insertions, updates and deletions of data are not efficient, that is why they are suitable for data warehouses.
9. *Support for declarative processing*: In data management, the advantages of a declarative language like SQL over a procedural approach are very well known. This is a clear motivation for trying to apply data management technologies in some application domains where procedural solutions are dominant. The management of sampled observation data in environmental applications is one such domain.
10. *Support for aggregation*: Aggregation functionalities through statistical methods are at the kernel of OLAP and must be supported to effectively perform observation data analysis.
11. *Support for iterative processing*: Recursive queries are required in only few data management applications. This is the reason why such functionalities were out of the scope of first SQL implementations. Current ISO SQL standard and DBMSs vendors support a kind of limited recursion. Regarding the analysis of observation data in environmental applications, such functionalities are commonly required to perform many simulations. Examples of these are forest fire propagation, oil spills, flooding, etc. Thus, although it is not a kernel functionality, the support for iterative processing is a desirable feature.
12. *Data processing based on a well known language*: As in the case of the data model, the definition of query languages that are based on well known ones as SQL is a clear advantage.

	Obs. Semantics	Sampled Data	Multiresolution	Simple Model	Well Known Model	Stream Proc.	OLTP	OLAP	Declarative Proc.	Aggregation	Iterative Proc.	Well Known Lang.	Impl. Available
OGC SWE Stds	Y	Y			Y								Y
Obs. Data Models	Y			Y	Y								
GIS		Y		Y	Y								Y
Sensor Stream				Y	Y	Y			P	P		P	Y
Spat. and ST DBMSs		Y			Y		Y	Y	P	P	P	P	Y
Spatial NoSQL							Y		Y				Y
Spatial HP DW				Y	Y			Y	P	P		P	Y
Array Data Managers		Y		Y				Y	Y	Y			Y
SciQL		Y			P			Y	Y	Y		P	Y
SODA	Y	Y	Y	Y				Y	Y	Y			

Fig. 2 Comparison of related approaches and technologies

13. *Availability of efficient implementation*: A data management approach is really useful if it can be efficiently implemented. A prototype implementation demonstrates the viability of the approach and its use in real application domains shows its matureness.

Based on the above evaluation criteria various research approaches and available technologies are now classified and qualitatively compared, including also the present SODA framework. An overview of such comparison is given in the table of Fig. 2, where each approach is marked with "Y" or "P" if it, respectively, supports or partially supports the relevant criterion. A more detailed discussion is given below.

2.1 OGC SWE standards

The Sensor Web Enablement (SWE) of the Open Geospatial Consortium (OGC) provides a series of standards for the interfaces of web services related to the management of environmental observation data. In particular, the Observations and Measurements (O&M) [1] and Sensor Model Language (SenorML) [2] where already mentioned in the introduction. The Sensor Observation Service (SOS) [3] defines an web service interface to query observation data collections, either stored or directly obtained from the devices. Data is transfered between client and server in standard XML encodings of O&M and SensorML models. Query capabilities of SOS are limited to just filtering. Regarding data processing, OGC defines the Web Processing Service (WPS)

[26] interface that enables the invocation of data processing algorithms through the web. Various implementations of the above standards exist already in the market, both with commercial and open source licenses. In general it is obvious that O&M provide appropriate support for the modeling of observation semantics and sampled data. Different spatial and temporal resolutions are supported but transformations are a user matter. The underlying object oriented data modeling approach with XML encodings is well known, however, to support sampled data nested structures are required. Declarative data processing is not supported at all as WPS just provides means for remote procedure calls.

2.2 Observation data models

Beyond the above O&M OGC standard, various data models and ontologies have been proposed to support observation data semantics [4–6]. They are based on well known paradigms and provide observation data semantics with simple data modeling approaches. However, sampled data and multi-resolution is out of the scope of these models as it is also any kind of data processing.

2.3 Geographic information systems (GIS)

Currently, a wide variety of GIS tools, both with commercial and open source licenses, are available. A representative example of them is GRASS [7], which supports the management of any kind of geographic data, including Rasters, recorded in many different well known models and formats. Raster data management is usually formalized with relevant Raster algebras [27]. Observation semantics are not considered in GIS and although the managed data may have many different spatial resolutions, transformations between them have to be explicitly done by the user to perform operations. Spatial data processing is a strength of tools like GRASS, however, it is performed by the execution of a very large amount of different commands, therefore, a declarative language is missing. Notice that the user must known which is the functionality of each command and how to combine them, thus only real experts may take real advantage of spatial data analysis with GIS tools.

2.4 Sensor stream processing approaches

Various Stream Processing approaches have been explicitly proposed for the management of data generated by sensor networks [8,9]. Despite of being defined for sensor data, observation data semantics are not explicitly incorporated by the system and are delegated to user interpretation. Any kind of spatial data management is out of the scope of these approaches. They support declarative real-time processing of streams with aggregation functionality based on SQL like languages. Real-time requirements of these approaches are clearly in conflict with the support of iterative processing.

2.5 Spatial and spatio-temporal DBMSs

Many temporal extensions have been proposed for the classical relational model [28, 29]. Recently, some characteristics have been incorporated into ISO SQL standard [30]. Various spatial [10,11,31] and spatio-temporal [32,33] extensions to classical models have been proposed in the literature. Spatial functionality has already been added to ISO SQL standard [12], which is currently implemented by most of the available DBMSs (see [13] for an example). The direct support of observation semantics is out of the scope of spatial DBMSs. They support the management of conventional E/R data with a well known object-relational paradigm and SQL, where properties of entities might have spatial data types (point, line, surface, etc.). Extensions for Raster data are also supported by some approaches and systems [13], however, they require nested structures and do not provide explicit support for multi-resolution. They can be used both for OLTP and OLAP, but in the general case they were not designed with Big Data requirements in mind. Regarding declarative data processing, it is only efficiently supported for non-sampled data and it includes both aggregations and SQL recursion for iterative queries. To manipulate raster data with SQL constructors it has to be unnested from a complex value of a raster data type, which is a highly inefficient task.

2.6 Spatial NoSQL approaches

Systems following a NoSQL approach and providing spatial data management capabilities are still few. An example is the extension of MongoDB [14] with support for the management of GeoJSON encoded data. Their functionality is very limited both in data modeling and processing.

2.7 Spatial high performance data warehouse approaches

To the best of these authors knowledge, only the Monetdb DBMS [15] provides spatial functionality on top of a high performance column-oriented implementation for OLAP. Neither observation semantics nor sampled data are directly supported by the system. Therefore, declarative processing is only partially supported. It lacks recursive queries, therefore iterative processing is not supported.

2.8 Array data managers

The management of sampled data fits very well array data management approaches. Currently, array algebras like the one in [34] are the basis for the development of relevant array data managers [16,17]. These systems use a simple array data model to provided high performance OLAP over very large arrays. They provide declarative array query languages that include aggregation capabilities. Iterative processing is not supported. Although the flavor of the languages is similar to that of SQL, both the array semantics and complex array operators make them quite cumbersome for DBMS

users. Observation semantics are out of the scope of these systems and multi-resolution is not explicitly supported.

2.9 SciQL

A SQL-based query language for science applications SciQL is defined in [18]. The language enables the integrated analysis of relational and array data, therefore both *Entities* and sampled data are supported. Obviously, declarative query with aggregation is supported, however, current implementation with MonetDB [15] technologies does not include recursion for iterative processing support. The incorporation of an array data structure adds complexity to the relational model and the extension of SQL with array semantics makes it less friendly to DBMS users. Observation semantics are out of the scope of the approach and specific support for multi-resolution in the type system is not provided.

2.10 SODA

The SODA framework described in the current paper is also added to the comparison for qualitative evaluation purposes. As it is shown, SODA supports both observation semantics and sampled data. It combines a multi-resolution type system with a simple non-nested data structure based on the well known mathematical concept of function. Declarative spatio-temporal analysis is supported and aggregation functionality is also incorporated. Support for iterative processing functionality and the efficient implementation of SODA are part of future work. Both the data model and query language of SODA are not based on the well known relational paradigm, which is somehow a setback for current DBMS users. However, the formalism is simple and it is based on the well known mathematical concept of function. The declarative language combines logical and functional constructors very similar to those already present in languages like XQuery. These characteristics will be described throughout the paper.

3 Observation data warehouse

This section describes the observation data warehouse that supports observation data recording. The general functionalities of an observation data management system briefly described in the introduction give rise to the following requirements for an observation data warehouse.

1. The model must support the representation of classical E/R data integrated with temporal, spatial and spatio-temporal sampled data.
2. The model must provide direct support for observation semantics, through the representation of appropriate required metadata, including *Processes* and *phenomenonTime*.
3. The type system of the model must provide support for the representation of temporal and spatial data at different resolutions. Besides, the transformation between

those resolutions must also be simplified with relevant implicit and explicit type castings.

Based on the above requirements, an underlying spatio-temporal data model is first defined and on top of it structures for observation metadata are added to achieve the final data model for the observation data warehouse.

3.1 Underlying spatio-temporal data model

As it was already shown in the previous section, supporting sampled data with a relational formalism leads to highly inefficient approaches. On the other hand, the use of functional models for the management of E/R data has already been tried in the area of Functional Databases [35]. The proposed spatio-temporal data model is based on the well known mathematical concept of function. It consists of conventional, temporal and spatial data types, functions to manipulate data values called *Intensional Mappings* and functions to record data values called *Extensional Mappings*. The recording of data singletons, called *Constants* is also supported by the model.

3.1.1 Data types

Conventional data types consist of *Boolean*, *CString* (variable size character strings), *Integer* and two floating point numeric types, *Float* and *Double*. Parametric type *FixedPrecision(P, S)* enables the user to specify precision (P, maximum number of decimal digits used) and Scale (S, number of digits in the fractional part) in a fixed point numeric representation. Default and also maximum values for P and S are defined by the system. We shall denote those values as DP (default P), MP (maximum P), DS (default S) and MS (maximum S). Any value N of type *FixedPrecision(P, S)* can be written in the form $n * 10^{-S}$, where n is an integer in the range $(-10^P, 10^P)$. All the above data types include a special *Undefined* value denoted by $\perp$.

Temporal data types The following three data types enable the modeling of the discrete multi-resolution time values.

Definition 1 Let R (resolution) be a value of data type *FixedPrecision*(P, S). Then the following temporal data types are defined.

TimeInstant(R):

$$\{t * R \mid t \epsilon \, Integer \wedge -10^{MP} < t < 10^{MP}\} \cup \{\perp\}$$

Time(R):

$$\{t * R \mid t \epsilon \, Integer \wedge 0 <= t * R < 24hours * 3600seconds/hour\} \cup \{\perp\}$$

Date(R):

$$\text{TimeInstant}(86400)$$

Time instant values are semantically interpreted as temporal shifts (positive or negative) in seconds from a reference system defined time instant ($t = 0$). A common value for such a reference time instant in current DBMS implementations is "1970-01-01 T 00:00:00.000000".

Spatial data types The following two data types enable the modeling of 1D and 2D Euclidean spaces.

Definition 2 Let P (Precision) and R (Resolution) be values of types *Integer* and *FixedPrecision*(PR, SR), respectively. Then the following spatial data types are defined.

Point1D(P, R):

$$\{x * R \mid x \epsilon\, Integer \wedge -10^P < x < 10^P\} \cup \{\perp\}$$

Point2D(P, R):

$$\{(x * R, y * R) \mid x, y \epsilon\, Integer \wedge -10^P < x, y < 10^P\} \cup \{\perp\}$$

Geometric data types Based on data type *Point2D*(P, R) and on the standard specification given by [12], the following data types enable the modelling of geometries in 2D Euclidean spaces.

- *LineString*(P, R): Vector polylines defined by sequences of elements of *Point2D*(P, R).
- *Polygon*(P, R): Vector polygons, possibly with holes, whose borders are defined by sequences of elements of *Point2D*(P, R).
- *GeometryCollection*(P, R): Heterogeneous collections of geometries of any of the data types *Point*(P, R), *Polyline*(P,R) and *Polygon*(P,R).
- *MultiPoint*(P, R): Homogeneous collections of geometries of type *Point2D*(P,R).
- *MultiLineString*(P, R): Homogeneous collections of geometries of type *LineString* (P,R).
- *MultiPolygon*(P, R): Homogeneous collections of geometries of type *Polygon*(P,R).
- *Geometry*(P, R): Abstract type. Geometries or collections of geometries of any of the above 2D types.

3.1.2 Intensional mappings

Informally, *Intensional Mappings* are functions defined over the available data types. These functions are always defined intensionally, in a mathematical sense, by either some algorithm or expression. Notice the difference with *Extensional Mappings* defined below in Sect. 3.1.3.

Definition 3 If $T_1, T_2, \ldots, T_n$ is a possibly empty sequence of not necessarily distinct data types and T is also a data type, then an *Intensional Mapping* with signature M(T_1, $T_2, \ldots, T_n$):T is defined as a function M: $T_1, T_2, \ldots T_n \rightarrow T$.

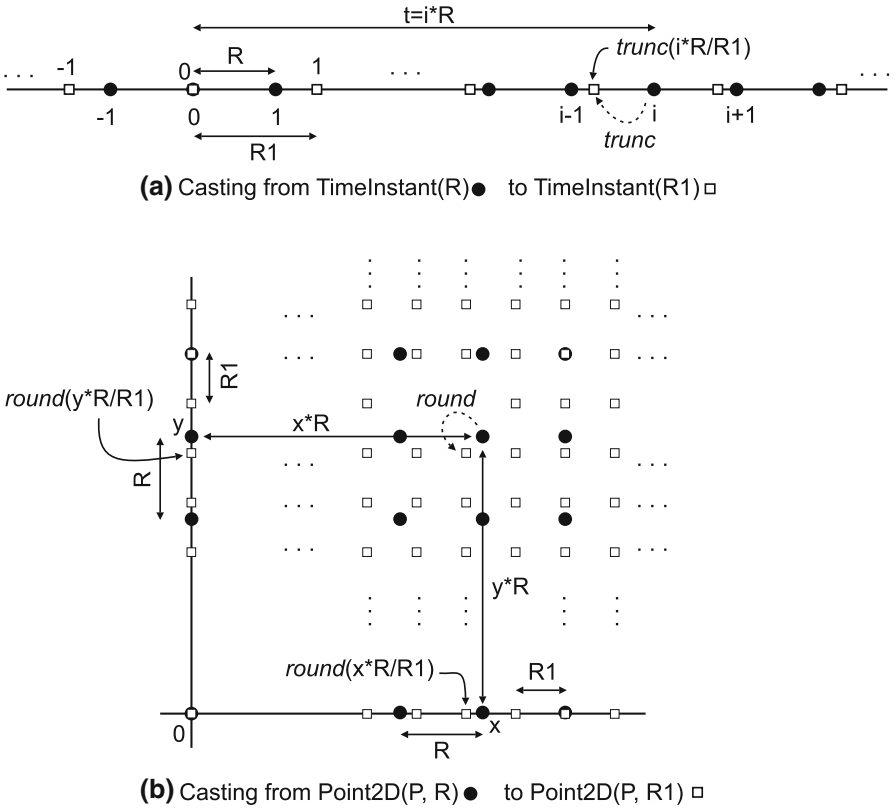

Fig. 3 Type castings between temporal and spatial types

Primitive *Intensional Mappings* are directly provided by the system in the syntactic form of operators, functions and type castings. Implicit type castings are applied between types of the same category (Boolean, CString and Numeric) during the evaluation of functions and operations.

Temporal intensional mappings Let $t = i * R, t_1 = i_1 * R$ be two values of type *TimeInstant*(R) and let $n = i_n * 10^{-S_R}$ be a value of type FixedPrecision(P_n, S_R), where S_R matches the scale of the FixedPrecision type of $R = r * 10^{-S_R}$. Then the expression $t - t_1$ yields value $(i - i_1) * R$ of type FixedPrecision(MP, S_R). The expression $t - n$ yields the *TimeInstant*(10^{-S_R}) value $(i * r - i_n) * 10^{-S_R}$. The expression $t + n$ yields the *TimeInstant*(10^{-S_R}) value $(i * r + i_n) * 10^{-S_R}$. A complete ordering is straightforwardly defined for values of each temporal data type and relevant comparison operations might be applied accordingly. Temporal functions are also provided although their definition is out of the scope of this paper. Castings between temporal types enable multi-resolution temporal management. Thus, the expression "*cast*(t to TimeInstant(R1))"yields the *TimeInstant*(R1) value $trunc(i * R/R1)$, as it is illustrated in Fig. 3a. Type castings are implicitly applied between temporal types during the evaluation of functions and operations.

Spatial intensional mappings If $p = (x * R, y * R)$, $p_1 = (x_1 * R, y_1 * R)$ are two values of type *Point2D*(P, R), then the expressions $p + p_1$ and $p - p_1$ yield respectively values $((x + x_1) * R, (y + y_1) * R)$ and $(((x - x_1) * R, (y - y_1) * R))$, both of the same Point2D(P, R) data type. For completeness, a total ordering is defined for Point2D data types, based on some space-filling curve [36]. Such an ordering enables the application of comparison operators. If x, y are values of type *FixedPrecision*(P,S), then function *point2d*(x, y) yields the value (x, y) of type Point2D(P, 10^{-S}). Inversely, functions *xcoord*(p) and *ycoord*(p) yields respectively the values $x * R$ and $y * R$ of type *FixedPrecision*(P, S_R), where S_R is the scale value of the *FixedPrecision* type of R. Other spatial functions are provided, including *distance* and *direction*, however, their definition is out of the scope of the paper. Type castings are defined between *Point2D* types that enable transformations between different spatial resolutions. Thus, the expression "*cast*(p to *Point2D*(P_1, R_1))" yields value (*round*($x * R/R_1$), *round*($y * R/R_1$)), as it is illustrated in Fig. 3b. Similar operations, functions and castings are also provided for *Point1D* types. Type castings are automatically applied between spatial types during the evaluation of operations and functions.

Geometric intensional mappings Geometric functions specified in [12] are also supported. Besides, type castings enable the transformation between geometric values of different spatial resolutions.

3.1.3 Extensional mappings

Informally, an *Extensional Mapping* is a function that maps values from a finite domain, defined as a finite subset of the Cartesian Product of data types, to a codomain defined by a data type. Components of the domain of an Extensional Mapping are called *Dimensions*. Informally, a *Dimension* is just a set of values of a given data type.

Definition 4 A *Dimension* d over data type T, denoted by d(T), is defined as a non-empty finite subset of $T - \{\perp\}$

Finite sets of values of a given data type are recorded in *Dimensions*. Temporal and spatial samplings, defined below, are specific types of *Dimensions* of special interest for spatio-temporal data representation.

Definition 5 Let min and max, $min < max$, be two values of the same *TimeInstant*, *Time*, *Date* or *Point1D* type T. Then a *1D Sampling* S from min to max, denoted by S(min, max) is defined as the following *Dimension* over data type T.

$$\{s \in T \mid min <= s <= max\}.$$

Definition 6 Let $s_m = (x_m, y_m)$ and $s_M = (x_M, y_M)$, $x_m < x_M$ and $y_m < y_M$, be two values of the same *Point2D* type T. Then a *2D Sampling* S from s_m to s_M, denoted by S(s_m, s_M) is defined as the following *Dimension* over data type T.

$$\{(x, y) \in T \mid x_m <= x <= x_M \wedge y_m <= y <= y_M\}.$$

Efficient physical structures avoid having to record the values of very large samplings (see Sect. 5.2).

Definition 7 If $d_1, d_2, \ldots, d_n$ is a sequence of not necessarily distinct *Dimensions* and T is a data type, then a *Extensional Mapping* with signature M($d_1, d_2, \ldots, d_n$): T is defined as a function M: $d_1, d_2, \ldots, d_n \rightarrow$ T. The definition of a *Extensional Mapping* is represented by a finite set of pairs (d, c), where d is an element of $d_1 \times d_2 \times \ldots \times d_n$ and c is an element of T.

Extensional Mappings in the proposed model have an extensional mathematical definition, since each valid combination of domain and codomain values is explicitly recorded in the model. Notice the difference with *Intensional Mappings*, whose definition is not given by an exhaustive listing of elements.

Informally, a *Constant* is just one value of a given data type.

Definition 8 A *Constant* C of type T, denoted by C:T is defined as an atomic value of type T.

Following a functional database approach [35], *Dimensions* and *Extensional Mappings* enable the modeling of Entities and Relationships between them. For example, student, courses and enrollment data may be modeled as follows.

Dimensions
StudentId(Integer)
CourseId(Integer)
Extensional Mappings
StudentName(StudentId):CString
CourseName(CourseId):CString
EnrollMentDate(StudentId, CourseId):Date

Beyond classical ER data, the model elegantly integrates the representation of scientific and sensor data. As an example, the evolution of sea surface temperature and salinity data over time (sampled every minute), depth (sampled every meter) and space (sampled every kilometer) might be modeled as follows:

Constants
minTime:TimeInstant(600), maxTime:TimeInstant(600),
maxDepth:Point1D(1),
southWestCorner:Point2D(1000), northEastCorner:Point2D(1000)
Dimensions
Time(minTime, maxTime)
Depth(*cast*(0 to Point1D(1)), maxDepth)
Space(southWestCorner, northEasCorner)
Extensional Mappings
Temperature(Time, Depth, Space):FixedPrecision(5, 2)
Salinity(Time, Depth, Space):FixedPrecision(5, 2)

The above data model fits very well column-oriented DBMS implementation techniques, which are currently the trending technology for the implementation of data warehouses (see Sects. 5.2 and 5.3).

3.2 Observation data model

Observed *Entities* and samplings are modeled in the proposed model with the concept of *Feature*. As in the E/R model, *Features* are classified into *Feature Types*. Both key and non-key *Properties* of *Features* may be defined. *Properties* of features may be observed by *Processes*.

Processes are also classified into *Process Types* and *Properties* of them may also be defined. For our purposes, a *External* process is performed out of the scope of the framework and the observations that it produces are loaded into the data warehouse with a typical ETL task. On the other hand, an *Internal* process is executed by the framework during the ETL task to produce observations derived from both the data that are being loaded and the data already available in the data warehouse. For our purposes, we shall classify processes both as either *Time-triggered* or *Event-triggered* and as either *External* or *Internal*.

The above concepts and the relationships between them are depicted in the UML class diagram of Fig. 4. The schema of a spatial observation data warehouse is defined in SODA using a XML language called XODDL (XML Observation Data Definition Language), which is defined in the present work as an XML encoding of the UML class diagram of Fig. 4.

In order to use the framework, the administrator must create a dataset scheme using XODDL. Next, *Observation Process* metadata is added to the framework. Internal *Processes* have also to be defined (see Sect. 4.2). Next, Feature Types with their non-observed properties are inserted. At this stage, the framework is ready to execute observation data ETL tasks. Each execution appends new values stamped with appropriate temporal data (*phenomenonTime*) to *Observed Properties* of *Feature Types*. Notice that new data is always appended and deletions and updates are not supported (except for data administration purposes).

The instances of the above concepts that define a proposed running example are depicted in the UML object diagram of Fig. 5. A more detailed description of *Feature* and *Process Types* and relevant *Dimensions* and *Extensional Mappings* used to record their data is given below.

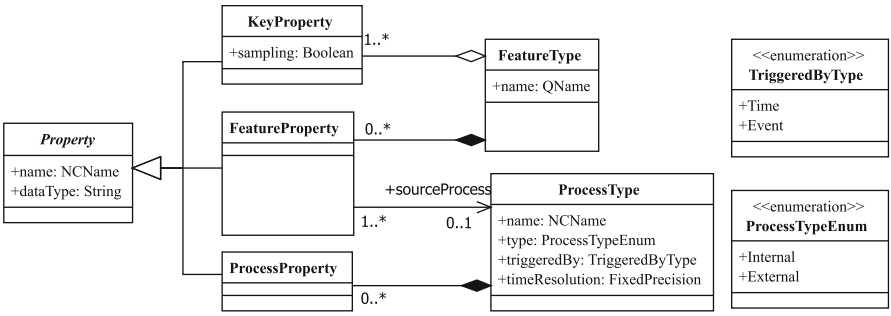

Fig. 4 UML class diagram of the observation data model

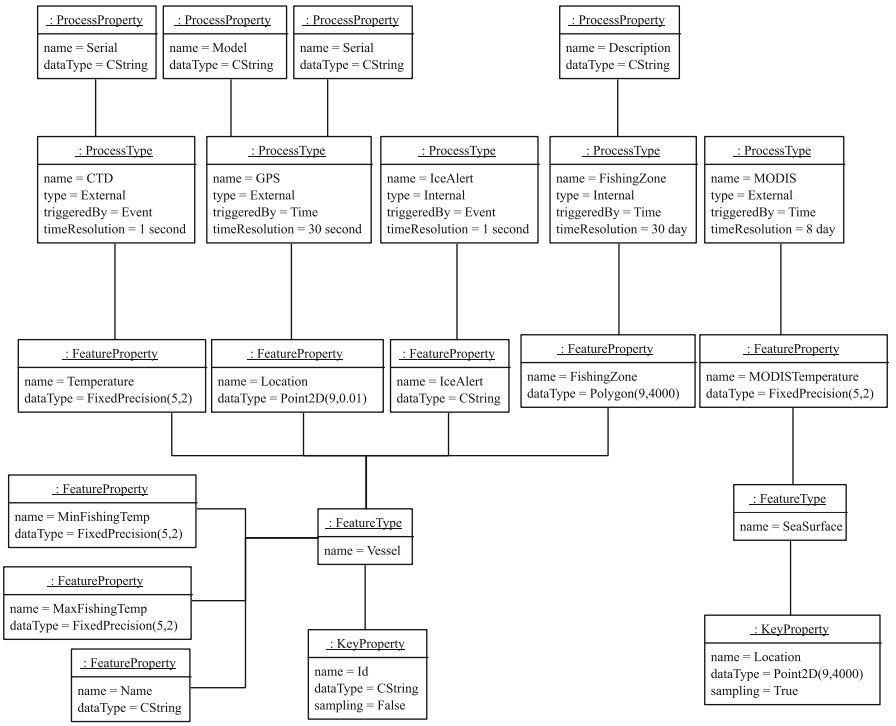

Fig. 5 UML object diagram of a running example

3.2.1 Feature types

Feature Types enable the integrated modeling of entities and samplings. Two *Feature Types* are defined in the running example, Vessel is used to model fishing boats (entities) and SeaSurface is used to model a geographic sampling. A *Feature Type* is defined by one or more *Key Properties* and zero or more *Feature Properties*. *Key Properties* are used to model either key properties of entities or dimensions of samplings. *Feature Properties* are used to model either non-key properties of entities or sampled properties. In the running example, vessels are identified by a Vessel.Id key property of type string, whereas the spatial dimension of SeaSurface is modeled by key property SeaSurface.Loc of type Point2D(9, 4000). Notice that an attribute (sampling="true") is added to the SeaSurface.Loc key property to indicate that it is a sampling dimension. Non-key properties of Vessel enable the recording of its name, minimum and maximum fishing temperatures, location, water temperature measured by the boat, ice alert risk according to measured temperature and the spatial fishing zone according to the fishing temperature interval. SeaSurface is a sampling that has just one MODISTemperature property that records the temperature at each location provided by MODIS sensor installed on board of Terra and Aqua NASA satellites. When the values of a specific *Feature Property* are generated by some Process, the relevant *Process Type* metadata is referenced by attribute "sourceProcessType". Notice

that in the running example, only names and fishing temperatures of vessels are not generated by Observation Processes.

The following *Dimensions* and *Extensional Mappings* are recorded in the framework for each *Feature Type* F.

- For each *Key Property* KP of type KPT of F, if the attribute sampling is false then a non-sampling *Dimension* F.KP(KPT) is recorded, otherwise, a sampling *Dimension* F.KP(lo, hi) is recorded, where lo and hi of type KPT are the values that define the boundaries of the sampling recorded in the framework. In the running example, dimensions
  Vessel.Id(CString) and
  SeaSurface.Loc(lo:Point2D(9,4000),hi:Point2D(9, 4000))
  enable the recording of relevant key properties. Notice that in the framework, each Dimension and Extensional Mapping must have a unique name, therefore, to avoid conflicts, the name of the *Feature Type* (or *Process Type*) is added as a prefix to the name of the relevant property.
- For each *Feature Property* FP of type FPT of F, such that FP is not generated by any source *Process*, an *Extensional Mapping*
  F.FP($F.KP_1$, $F.KP_2$, ... $F.KP_n$):FPT
  is recorded, where each $F.KP_i$ is a *Dimension* corresponding to a relevant key property of F. In the running example, *Extensional Mappings*
  Vessel.Name(Vessel.Id):CString,
  Vessel.MinFishingTemp(Vessel.Id):FixedPrecision(5, 2) and
  Vessel.MaxFishingTemp(Vessel.Id):FixedPrecision(5, 2)
  enable the recording of relevant non-key feature properties not generated by *Observation Processes*.
- For each *Feature Property* FP of type FPT of F, such that FP is generated by an observation source process of type SP and KP_1, KP_2, ... KP_n are key properties of F:
  1. An *Extensional Mapping*
     F.FP($F.KP_1$, $F.KP_2$, ... $F.KP_n$, SP.Time):FPT
     enables the recording of the generated observations.
  2. An *Extensional Mapping*
     F.FP.Process($F.KP_1$, $F.KP_2$, ... $F.KP_n$, SP.Time):CString
     enables the recording of the specific *Observation Process* used at each time instant to generate observations.

  Thus, for example,
  Vessel.Location(Vessel.Id, GPS.Time):Point2D(9, 0.01)
  enables the recording of the evolution with respect to time of the vessel location generated by the GPS devices. The identifier of the GPS device used at each specific instant in a given vessel is provided by *Extensional Mapping*
  Vessel.Location.Process(Vessel.Id, GPS.Time):CString.

3.2.2 Process types

Process Types record metadata of referenced observation *Process*. A *Process Type* may be either *Time* or *Event-Triggered* at a given time resolution. Thus, for example, GPS processes of vessels generate vessel location observations every 30 seconds. On the other hand, at any moment the vessel crew uses a CTD sensor to obtain a temperature observation, whose time is recorded with a resolution of 1 second. The above processes are external to the framework. An example of an internal process is the IceAlert that computes an ice risk value (either "High" or "Medium") every time the CTD sensor generates an observation with a temperature below zero. Internal processes are executed by the framework during each ETL task to generate calculated *Feature Properties*. Thus, the FishingZone property of each vessel (Polygon2D type) is generated from sea surface temperatures provided by MODIS. The capabilities of SODA to define internal analytical processes are described in Sect. 4.2. Finally, each process of a given *Process Type* might be described by a set of *Process Properties*. Thus, for example, a GPS process has Model and Serial properties as part of its metadata.

The following *Dimensions* and *Extensional Mappings* are recorded in the framework for each *Process Type* P.

- A dimension P(CString) enables the recording of all the identifiers of processes (instances) of P. In the running example, dimensions
  GPS(CString)
  CTD(CString)
  IceAlert(CString)
  FishingZone(CString) and
  MODIS(CString)
  enable the recording of all the required process identifiers.
- For each *Process Property* PP of type PPT, an *Extensional Mapping*
  P.PP(P):PPT
  is recorded. Thus, *Extensional Mappings*
  GPS.Model(GPS):CString and
  GPS.Serial(GPS): CString
  enable the recording of GPS models and serial numbers in the running example. Similarly, *Extensional Mappings*
  CTD.serial(CTD):CString and
  FishingZone.Description(FishingZone):CString
  enable the recording of CTD serial numbers and FishingZone internal Process descriptions.
- If P is a time-triggered Process of resolution R then a *Sampling*
  P.Time(lo:TimeInstant(R), hi:TimeInstant(R))
  is recorded, where lo and hi are respectively the lowest and highest time instants inserted in the framework for observation times generated by P. On the other hand, if P is an event-triggered Process of resolution R then a non-sampling *Dimension*
  P.Time(TimeInstant(R))
  is recorded.

4 Spatial observation data analysis

The following requirements for spatial observation data analysis are derived from the generic functionalities of an observation data management system given in the introduction.

1. The framework must support OLAP over large warehouses of spatial observation data.
2. The framework must support the definition of the behavior of *Internal Processes* using a spatio-temporal declarative analysis language.
3. The language must support the integrated analysis of both conventional E/R data and temporal, spatial and spatio-temporal sampled data.
4. The language must support aggregation functionality.

Based on the above requirements, first an XML Mapping Analysis Language (MAPAL) is defined and next it is shown how it can be used for the definition of internal analytical observation processes that are executed by SODA during ETL.

4.1 Mapping analysis language

The use of data model based on functions and not on sets or sequences disables the direct use of some extension of well known languages like SQL and XQuery. However, the hybrid logical-functional paradigm of the proposed MAPAL language reuses constructs of those well known languages and combines them with an XML syntax that simplifies their insertion in currently dominating web service interfaces. In particular, derived *Constants* and *Intensional* and*Extensional Mappings* are defined using three types of expressions, namely, *Functional Expressions*, *Conditional Expressions* and *Aggregate Expressions*. On the other hand, new *Dimensions* are defined using either *Sampling* or *Dimension Expressions*. The syntax and semantics of all those expressions are presented and illustrated with examples below.

Functional expressions A functional expression e defined in the context of variables $v_1, v_2, \ldots, v_n$, denoted by $e(v_1, v_2, \ldots, v_n)$, combines context variables with already defined constructors, system provided literals, operators, primitive mappings and type castings. The semantics are the obvious ones. The example below illustrates the use of functional expressions to define two *Constants* and a *Extensional Mapping*.

Example 1 Obtain an extensional mapping that determines for each time instant whether vessel “Bur124” is navigating inside or outside its fishing temperature interval.

```
<Constant name="MinBur124">
 <Return>Vessel.MinFishingTemp("Bur124")</Return>
</Constant>

<Constant name="MaxBur124">
 <Return>Vessel.MaxFishingTemp("Bur124")</Return>
</Constant>

<ExtensionalMapping name="InsideTemperature" domain="GPS.Time t">
<Return>
 SeaSurface.MODISTemperature(t,Vessel.Location(t,"Bur124"))&gt;=MinBur124
AND SeaSurface.MODISTemperature(t,Vessel.Location(t,"Bur124"))&lt;=MaxBur124
</Return>
</ExtensionalMapping>
```

Notice that functional expressions used to define constants do not have context variables. On the other hand, the functional expression used to define *Extensional Mapping* "InsideTemperature" has a context variable t whose scope is the dimension GPS.Time of its domain. Notice also that symbols $<$ and $>$ are not allowed in XML content and their entities < and > have to be used instead.

Conditional expressions They enable the introduction of if-then-else structures in the evaluation of *Constants*, *Intensional* and *Extensional Mappings*. The semantics are the obvious ones and the syntax is illustrated below with an example.

Example 2 Define an Intensional Mapping SeaIce(t, p) that yields the risk of having ice in point p at time instant t. Risk "Red" for temperatures below -2 degrees, risk "Orange" for temperatures between 0 and -2 and risk "Green" for temperatures above 0 degrees.

```
<IntensionalMapping name="SeaIce" domain="t, p">
  <When>fish:SeaSurface.MODISTemperature(t, p)&lt; -2 </When>
  <ThenReturn>"Red"</ThenReturn>
  <When>SeaSurface.MODISTemperature(t, p)&gt;= -2
    AND SeaSurface.MODISTemperature(t, p)&lt;= 0
  </When>
  <ThenReturn>"Orange"</ThenReturn>
  <ElseReturn>"Green"</ElseReturn>
</IntensionalMapping>
```

Set operations *Union* and *Intersection* between *Dimensions* are now formalized, as a prerequisite for the definition of the *Dimension* and *Aggregate Expressions* of MAPAL.

Definition 9 Let $d_1(T_1)$ and $d_2(T_2)$ be two non-sampling *Dimensions*, where T_1 and T_2 are compatible data types, i.e., an implicit casting has been defined among them. Then d_1 *Union* d_2 is defined as the *Dimension* $d(T) = \text{cast}(d_1 \text{ as T}) \cup \text{cast}(d_2 \text{ as T})$, where T is the result type of the implicit cast between T_1 and T_2 and cast(d_i as T) is the *Dimension* resulting from casting each element of d_i to type T.

Definition 10 Let $d_1(T_1)$ and $d_2(T_2)$ be two *Dimensions*, where T_1 and T_2 are compatible data types whose implicit result cast type is T, and at least one of the *Dimensions* is a *1D Sampling*. Then d_1 *Union* d_2 is defined as the *1D Sampling* S(m, M), where

$$m = min\{cast(v \text{ as } T) | v \epsilon\, d_1 \cup d_2\}$$
$$M = max\{cast(v \text{ as } T) | v \epsilon\, d_1 \cup d_2\}$$

Definition 11 Let $d_1(T_1)$ and $d_2(T_2)$ be two *Dimensions*, where T_1 and T_2 are compatible data types whose implicit result cast type is T, and at least one of the Dimensions is a *2D Sampling*. Then d_1 *Union* d_2 is defined as the *2D Sampling* S((x_m, y_m), (x_M, y_M)), where

$$x_m = min\{cast(x\ as\ T)|(x, y) \epsilon\ d_1 \cup d_2\}$$
$$y_m = min\{cast(y\ as\ T)|(x, y) \epsilon\ d_1 \cup d_2\}$$
$$x_M = max\{cast(x\ as\ T)|(x, y) \epsilon\ d_1 \cup d_2\}$$
$$y_M = max\{cast(y\ as\ T)|(x, y) \epsilon\ d_1 \cup d_2\}$$

Definition 12 Let $d_1(T_1)$ and $d_2(T_2)$ be two *Dimensions* (either sampling or non-sampling), where T_1 and T_2 are compatible data types whose implicit cast type is T. Then d_1 *Intersection* d_2 is defined as the *Dimension* $d(T)$ = cast(d_1 as T) ∩ cast(d_2 as T), where cast(d_i as T) is the *Dimension* resulting from casting each element of d_i to type T

Defining *Union* and *Intersection* in this way between samplings simplifies efficient implementation structures and matches the requirements of applications.

Aggregate expressions They are composed of the following four sections:

- *Variable scope specification*: It is composed of a sequence of variable scope specifications of the form

```
<ForEach var="v1">dimSetExpr1</ForEach>
<ForEach var="v2">dimSetExpr2</ForEach>
...
<ForEach var="vn">dimSetExprN</ForEach>
```

  where each vi is a variable name that iterates over the elements of the result of each $dimSetExpri$. Each $dimSetExpri$ is a *Dimension Set Expression* of the form $d1$ OP $d2$ OP ... OP dm, where each di is a *Dimension* name and OP is either AND or OR. The semantics of AND and OR are those of *Dimension Set Operations Intersection* and *Union*, respectively.
- *Filtering*: The valid combinations of variable values may be restricted with an expression of the form

```
<Where>c(v1, v2, ..., vn)</Where>
```

  where $c(v1, v2, \ldots, vn)$ is a *Functional Expression* that evaluates to boolean value true only for valid combinations of variable values. Valid combinations of variables are retrieved as a sequence of tuples, each of the form ($v1$, $v2$, …, vn), ordered ascending by $v1, v2, \ldots, vn$. A distinct ordering for the sequence may be optionally specified in the following *Ordering* section.
- *Ordering*: It is an optional sequence of expressions of either of the following two forms

```
<OrderAscendingBy>o(v1, v2, ..., vn)</OrderAscendingBy>
<OrderDescendingBy>o(v1, v2, ..., vn)</OrderDescendingBy>
```

  where o is a functional expression whose result is of some ordered type. The tuple sequence resulting from the above sections is now ordered either ascending or descending by the result of these functional expressions.

– *Aggregate Evaluation*: It specifies an aggregate expression of the form

```
<Aggregate>a(v1, v2, ..., vn)</Aggregate>
```

where *a* combines functional expression elements with system provided aggregate functions. Variables *vi* must appear in *a* inside the scope of some aggregate function. Aggregate functions provided by the system may have the form of the classical ones of SQL (AVG, SUM, etc.) but they may also exploit the sequence ordering. Thus for example, function ATPOSITION(S, n) yields element located at position n in sequence S.

The following example illustrates the use of an aggregate expression for the definition of an *Extensional Mapping*.

Example 3 Obtain the number of vessels that, at each time instant t, are navigating through areas that have some risk of ice.

Dimension expressions A new non-sampling *Dimension d* may be defined with an expression whose general form is as follows.

```
<Dimension name="d">
  <ForEach var="v1">dimSetExpr1</ForEach>
  <ForEach var="v2">dimSetExpr2</ForEach>
  ...
  <ForEach var="vn">dimSetExprN</ForEach>
  <Where>c(v1, v2, ..., vn)</Where>
  <Return>e(v1, v2, ..., vn) </Return>
</Dimension>
```

The semantics are similar to those of *Aggregate Expressions*, however tuple sequence semantics are now replaced by tuple set semantics (as in the relational model).

Example 4 Obtain a new dimension "boat3" that contains the names of the boats that at instant "2013-01-04T12:00:00" are navigating through a zone with temperature below 3 degrees.

```
<Dimension name="boat3">
  <ForEach var="v">Vessel.Id</ForEach>
  <Where>
   SeaSurface.MODISTemperature("2013-01-04T12:00:00",
                          Vessel.Location("2013-01-04T12:00:00",v)) &lt; 3
  </Where>
  <Return>Vessel.Name(v)</Return>
</Dimension>
```

Sampling expressions A new *Sampling Dimension* S(m, M) may be defined with an expression of the form

```
<Dimension name="S">
 <Start>m</Start>
 <End>M</End>
</Dimension>
```

4.2 Definition of analytical processes

An XML language based on the MAPAL language described in the previous subsection enables the administrator of SODA to declaratively define internal analytical

Observation Processes that are executed during ETL. Each internal process is of a specific Process Type, of those declared in XODDL. Therefore, during its definition, both a unique Process identifier and values for relevant Process Properties have to be provided. Besides, contrary to external processes, an internal process has a definition that is expressed with MAPAL. In particular, such a definition is composed of three sections.

1. *Preliminaries*: A possibly empty preliminary collection of temporary *Constant* and *IntensionalMapping* definitions that will be reused in the remainder sections.
2. *Time Dimension Definition*: Definition of the time *Dimension* of the result Process. It is recalled that each Process Type P of the framework has a *Dimension* P.Time, which is a *Sampling Dimension* in the case of time-triggered *Processes* and a non-sampling *Dimension* in the case of event-triggered *Processes*.
3. *Observed Property Definitions*: Definition of an *Extensional Mapping* with appropriate domain for each observed Feature Property whose values are generated by Processes of the present Process Type.

We shall focus here in points 2 and 3 above, since point 1 is just syntactic sugar. The time dimension of a Time-Triggered internal process type P of time resolution R is defined with an XML expression of the form

```
<TriggeredByTime>P1.Time, ..., Pn.Time</TriggeredByTime>
```

where each Pi.Time is the time *Dimension* of a Process Type Pi. The semantics are those of the *1D Sampling* S(m, M), where

$$\text{m} = \text{cast}(min\{v | v \in P1.Time \cup \ldots \cup Pn.Time\} \text{ as TimeInstant(R)})$$
$$\text{M} = \text{cast}(max\{v | v \in P1.Time \cup \ldots \cup Pn.Time\} \text{ as TimeInstant(R)})$$

The time dimension of an event-triggered internal process type P of time resolution R is defined with an XML expression of the form

```
<TriggeredByEvent>
  <Event var="t">P1.Time, P2.Time, ..., Pn.Time</Event>
  <Condition>c(t)</Condition>
</TriggeredByEvent>
```

where each Pi.Time is the time *Dimension* of a Process Type Pi and c(t) is a functional expression of boolean type. The semantics are those of the non-sampling *Dimension* defined by the set

$$\{cast(t \text{ as } TimeInstant(R)) | t \in P1.Time \cup \ldots \cup Pn.Time \wedge c(t)\}.$$

Each feature property F.FP observed by an internal process type P has to be defined as an *Extensional Mapping* within the definition of P. The first dimension of the domain of F.FP will be P.Time. During ETL, each such *Extensional Mapping* will be evaluated, but restricting the scope of the evaluation to the P.Time dimension elements to be imported. This avoids the re-evaluation of the *Extensional Mapping* for the whole time extension of the data warehouse.

The definition of process types “IceAlert” and “FishingZone” of the running example are given next for illustration purposes.

```
<?xml version="1.0" encoding="utf-8"?>
<pd:ProcessDefinitions
 xmlns="es.usc.citius.de.mapal"
 xmlns:pd="es.usc.citius.de.soda.ProcessDefinition">

 <pd:Process id="IceAlert" processType="IceAlert">
   <pd:Definition>
     <IntensionalMapping name="IceRisk" domain="temperature">
       <When>temperature &lt;= -2</When><ThenReturn>"High"</ThenReturn>
       <When>temperature &lt; 0</When><ThenReturn>"Medium"</ThenReturn>
     </IntensionalMapping>
     <IntensionalMapping name="ExistsVesselInRisk" domain="t">
       <ForEach var="v">Vessel.Id</ForEach>
       <Where>Vessel.Temperature(t, v) &lt; 0</Where>
       <Aggregate>not EMPTY(v)</Aggregate>
     </IntensionalMapping>
     <pd:TriggeredByEvent>
       <pd:Event var="t">CTD.Time</pd:Event>
       <pd:Condition>ExistsVesselInRisk(t)</pd:Condition>
     </pd:TriggeredByEvent>
     <ExtensionalMapping name="Vessel.IceAlert"
                         domain="IceAlert.Time t, Vessel.Id v">
       <Return>IceRisk(Vessel.Temperature(t, v))</Return>
     </ExtensionalMapping>
   </pd:Definition>
 </pd:Process>

 <pd:Process id="FishingZone" processType="FishingZone">
   <pd:Definition>
     <pd:TriggeredByTime>MODIS.Time</pd:TriggeredByTime>
     <ExtensionalMapping name="Vessel.FishingZone"
                         domain="FishingZone.Time t, Vessel.Id v">
       <ForEach var="p">SeaSurface.Loc</ForEach>
       <Where>SeaSurface.MODISTemperature(t, p) &gt;=
              Vessel.MinFishingTemp(v)
          AND SeaSurface.MODISTemperature(t, p) &lt;=
              Vessel.MaxFishingTemp(v)
       </Where>
       <Aggregate>VECTORIZE(p)</Aggregate>
     </ExtensionalMapping>
   </pd:Definition>
   <Description>
     This process obtains the piece of sea surface with
     appropriate water temperature for fishing.
   </Description>
 </pd:Process>

</pd:ProcessDefinitions>
```

First “IceAlert” is defined as an event-triggered Process that is fired every time a CTD measures a temperature below 0. The Vessel.IceAlert Feature Property is then defined as either “High” or “Medium” depending on the temperature value. Notice that a vessel might measure a temperature above zero at the same time another one measures a temperature below zero. In that case, *Extensional Mapping* “Vessel.IceAlert” would return an undefined value for the first vessel. The definition of “FishingZone” Process illustrates the use of aggregate function VECTORIZE to generate a vector polygon from the set of Point2D elements of a 2D *Sampling* for which a specific condition holds.

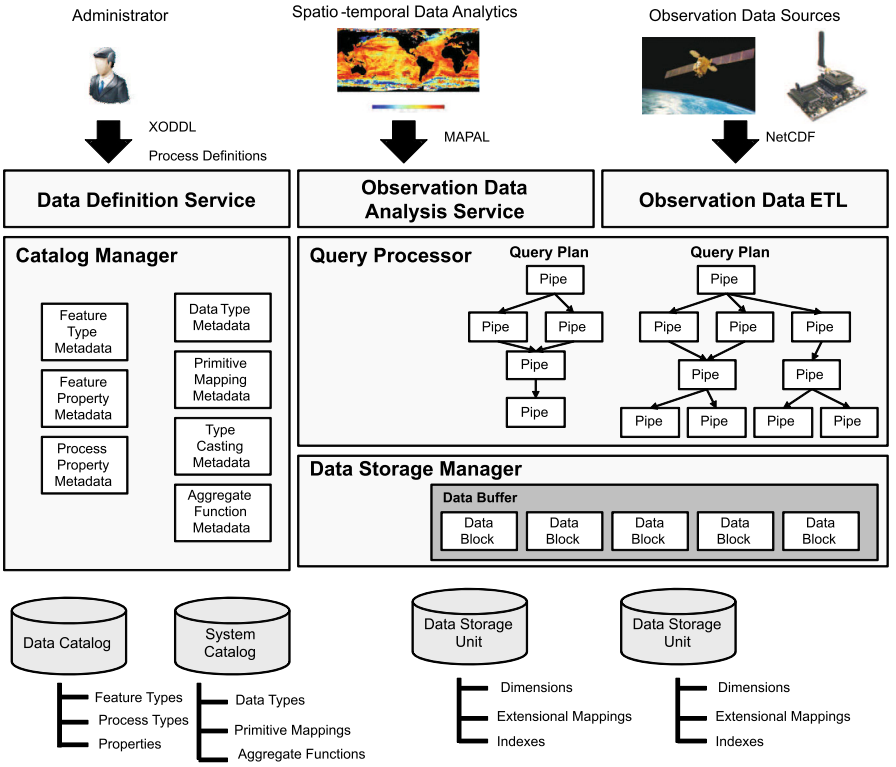

Fig. 6 Overview of SODA architecture

5 Column-oriented implementation issues

This sections discusses some issues that are guiding to the column-oriented implementation of SODA that is currently being undertaken.

5.1 SODA architecture

A general overview of the architecture of the framework is depicted in Fig. 6. At the bottom of the figure, on the left side data catalog and system catalog are shown. The former records metadata related to *Feature* and *Process Types* and their relevant *Properties*. The latter records metadata of system provided constructors, including data types, primitive mappings and operators, aggregate functions and type castings. *Dimensions* and *Extensional Mappings* are recorded in one or various *Data Storage Units*. Besides, appropriate indexes are also recorded to speed up direct access to the above data structures. Following common practice in column-oriented DBMS implementations, lightweight compression techniques will be adopted for physical data representation (see Sect. 5.2). The *Data Storage Manager* enables efficient data access to the storage units, implementing buffering of data blocks. Access to all the

system metadata is provided through the Catalog Manager. *Feature* and *Process* Metadata are generated from the schema definitions provided by the system administrator through the *Data Definition Service*. Besides the interpretation of XODDL (see Sect. 3.2) and internal process definitions (see Sect. 4.2), the Data Definition Service must enable the insertion of external processes together with their relevant Process Property values and non-observed feature data. Spatio-temporal data analytics are supported by an *Observation Data Analysis Service*, whose main functionality is the parsing and execution of MAPAL scripts. The execution of MAPAL query plans is performed by the *Query Processor*. Each query plan is represented by a *Direct Acyclic Graph* of pipes. Each pipe is generally executed in a distinct thread and supports a specific operation of the physical algebra described in Sect. 5.3. Finally, the *Observation Data ETL* component supports the execution of the internal analytical processes defined in the catalog during each observation data Extract Transform and Load (ETL).

5.2 Physical data representation

Various issues related to the physical data representation that are being considered during the current implementation are now discussed. Generally, each *Dimension* and *Extensional Mapping* is recorded as a header and a sequence of data blocks. *Dimension* data is recorded in self order whereas *Extensional Mappings* are recorded in the order of their respective *Dimensions*. Following common practices in column-oriented implementations, light weight compression techniques may be used to record both *Dimensions* and *Extensional Mappings*.

5.2.1 Light weight compression

The use of Light Weight Compression produces great performance improvement in current column-oriented implementations [37]. Thus, a requisite for the present SODA implementation is its flexibility to incorporate new compression techniques for both *Dimensions* and *Extensional Mappings*, enabling if possible the direct processing of compressed data. The current ongoing implementation uses three types of *DataBlock*, whose in-memory relevant data structures are depicted in Fig. 7 as a UML class diagram. A RLEBlock stores an element of a Run Length Encoding (RLE) sequence, i.e., it represents the repetition of a value from a start position to an end position. A RangeBlock represents a sequence of consecutive values of a data type by just recording the start and end values. A PlainBlock records a sequence of uncompressed values of a data type. the attribute "defined" of DataBlock records a compressed Bitmap that specifies which of the elements of RLEBlocks and RangeBlocks are valid.

A *Sampling* $S(c_1, c_2)$ dimension is recorded as a single *RangeBlock*, that records the boundaries c_1 and c_2 of S. Non-sampling *Dimensions* are recorded with sequences of self ordered *PlainBlocks*, i.e., non-compressed data. In the future, delta encodings will be added to support efficient compression of numeric non-sampling *Dimensions*. Regarding *Extensional Mappings*, they are recorded as sequences of *DataBlocks*, ordered by the dimensions of their domain. Either *RLEBlocks* or *PlainBlocks* are used to record *Extensional Mappings*. Other compression techniques including delta

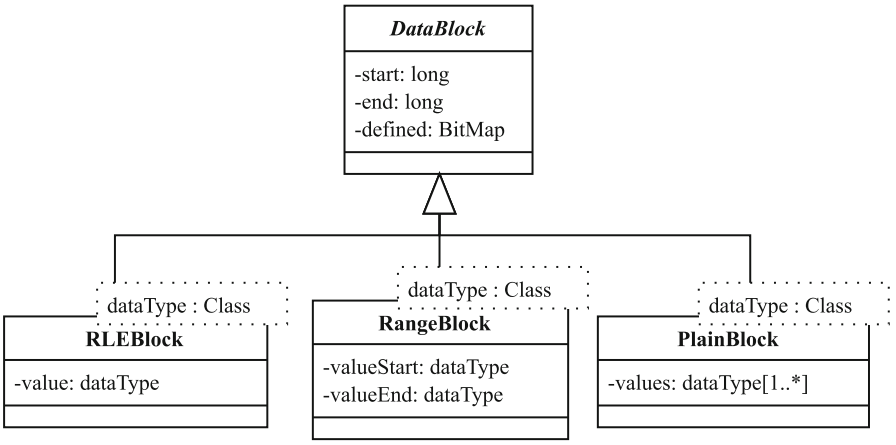

Fig. 7 Data block types in SODA

encodings and dictionary encodings will be added in the future. Properties *start* and *end* of a *DataBlock* are used in memory to reference the range of positions that represents the block inside its *Dimension* or *Extensional Mapping*. Property *defined* is used to determine which elements of the block are valid, i.e., which of them are distinct from *Undefined*. These *defined* bitmaps may be set as a consequence of the evaluation of a where clause of MAPAL.

5.2.2 Ordering

As it was already stated, *Dimensions* are self ordered whereas *Extensional Mappings* are ordered by the *Dimensions* of their domains. The time dimension of defined processes will always be the first *Dimension* of relevant *Extensional Mappings* of observed properties. This decision facilitates appending new temporal data without having to insert data values in the middle of already existing *Extensional Mappings*. *1D Samplings* are implicitly ordered from lower to higher values. Regarding *2D Samplings*, currently a linear ordering is used, however, in the future other orderings defined by other Space Filling Curves shall also be considered.

5.2.3 Standardized data interchange

The interchange of data between the framework and its data providers and data consumer is based on the use of the NetCDF standard file format. This format enables the efficient representation of array data and it is broadly used in meteorological and oceanographic applications. Currently it is also an OGC standard. Roughly speaking, data in a NetCDF is organized in dimensions and variables defined on those dimensions. These NetCDF concepts fit very well with the *Sampling* and *Extensional Mapping* concepts of the present framework.

5.3 Query processing

Issues related to the evaluation of MAPAL expressions are now discussed. In particular, first the physical algebra that is behind query processing is defined and next a couple of issues related to the current implementation of this algebra are discussed.

5.3.1 Physical algebra

The proposed physical algebra is a many-sorted algebra over elements of four different data structures, namely, *Dimensions*, *Domains*, *Constants* and *Extensional Mappings*. *Dimensions*, *Constants* and *Extensional Mappings* have already been defined in Sect. 3.1. A *Domain* is defined as either a *Dimension* or a Cartesian Product of 2 or more not necessarily distinct *Dimensions*. Operations of the algebra are classified into four different groups according to the result structure that they produce. In general, the signature of each operation has the form

OperatorName[ParameterList](ArgumentList)

where parameter list is a comma separated list of parameters and ArgumentList is a comma separated list of arguments. Arguments are denoted by different characters, possibly subscripted, depending on their type. Thus d_i, D_i, C_i and M_i denote respectively *Dimensions*, *Domains*, *Constants* and *Extensional Mappings*. Operations that produce *Dimensions* and *Domains* are described in Table 1. Table 2 describes operations that evaluate *Constants* and *Extensional Mappings*.

Table 1 Dimension and domain physical algebra operators

Dimension operators	
DimensionScan[dimName]()	Parameter *dimName* is the name of a *Dimension*. It generates a *RangeBlock* describing the range of positions in *dimName*
Union(d_1, d_2)	It performs the set *Union* of dimensions d_1 and d_2 (see Sect. 4.1)
Intersection(d_1, d_2)	It performs the set *Intersection* of dimensions d_1 and d_2 (see Sect. 4.1)
Sampling(c_1, c_2)	It produces a sampling dimension whose boundaries are defined by constants c_1, c_2. The result is encoded as a *RangeBlock*
DupRem(M)	It generates the new *Dimension* resulting from the elimination of duplicates from the codomain of *Extensional Mapping* M
Domain operators	
Domain(d)	It generates a *Domain* of just one component from dimension d
Product(D, d)	It performs the Cartesian Product between *Domain* D and *Dimension* d
Select(M)	If *Extensional Mapping* M has a codomain of boolean type, then it selects from the *Domain* of M the combinations where M yields true

Table 2 Constant and mapping physical algebra operators

Constant operators	
ConstantFetch[consName]()	It obtains from storage the value recorded for *Constant consName*
Literal[literal]()	It produces the value represented by *literal*
IntensionalMapping[iMap](C_1, C_2, …, C_n)	*iMap* is the name of either a primitive mapping, casting or operator provided by the system or an *Intensional Mapping* defined by the user. It evaluates *iMap* with arguments C_1, C_2, …, C_n
ExtensionalMapping[eMap](C_1, C_2, …, C_n)	*eMap* is the name of a *Extensional Mapping*. It obtains from storage the value of eMap for the domain element defined by C_1, C_2, …, C_n
Conditional(C_1, C_2, [C_3])	If C_1 is true then it yields C_2 otherwise, if C_3 is specified then it yields C_3, otherwise it yields *Undefined*
Aggregate[aFun](M_1, M_2, …, M_n, [MO_1, MO_2, …, MO_m])	All the argument *Extensional Mappings* are defined over the same *Domain*. It first produce a sequence of n-tuples from the codomains of M_1, M_2, …, M_n, ordered by the codomains of MO_1, MO_2, …, MO_m. Then it evaluates Aggregate Function AFun over such an ordered sequence
Mapping operators	
Constant(D, C)	It generates an *Extensional Mapping* that yields the result of evaluating *Constant* C for each element of Domain D
Project[dimRef](D)	For each element of Domain D it yields the value of *Dimension* referenced by *dimRef* of D
IntensionalMapping[iMap](M_1, M_2, …, M_n)	*iMap* is the name of either a primitive mapping, casting or operator provided by the system or an *Intensional Mapping* defined by the user. It evaluates *iMap* using as arguments the values obtained from the evaluation of M_1, M_2, …, M_n, for each element of their common *Domain*
ExtensionalMapping[eMap](M_1, M_2, …, M_n)	*eMap* is the name of a Extensional Mapping. For each element of the common *Domain* of M_1, M_2, …, M_n, it obtains from storage the value of *eMap*, using as domain the n-tuple resulting from the evaluation of (M_1, M_2, …, M_n)
Conditional(M_1, M_2, [M_3])	For each element of the common *Domain* of M_1, M_2 and M_3, if M_1 evaluates to true then it yields the result of evaluating M_2 otherwise, if M_3 is specified then it yields the result of evaluating M_3, otherwise it yields *Undefined*
Aggregate[groupBy][aFun](M_1, M_2, …, M_n, [MO_1, MO_2, …, MO_m])	All the argument *Extensional Mappings* are defined over the same *Domain*. The sequence of n-tuples resulting from the evaluation of M_1, M_2, …, M_n are grouped by dimension references *groupBy*. Each group of n-tuples is ordered by the result of evaluating MO_1, MO_2, …, MO_m. Finally, Aggregate Function AFun is evaluated in the scope of each group

5.3.2 Pipeline implementation

The physical algebra of the previous section is implemented following a *Producer-Driven Pipelining* approach. Thus, each operator is implemented as a pipe that is generally executed in a different thread. Data and reference interchange between pipes is asynchronous and it is supported by appropriate *DataBlock* buffers. This enables *Constant* and *Mapping* evaluation to be performed in parallel by different processing units, if they are available. Besides, if various storage units are available, different *Dimensions* and *Extensional Mappings* might be retrieved from storage also in parallel. This is known in the literature by pipeline parallelism [38]. The degree of pipeline parallelism is increased by the vertical partitioning approach followed by column-oriented implementations as the present one. An evaluation plan using the above operation pipes for the query of Example 3 is given in Fig. 8. First, two *DimensionScan* pipes obtain the range of reference positions of the two involved dimensions. Notice that with just one *RangeBlock* for each pipe, all the reference positions of both *Dimensions* are represented. After passing through *Domain* and *Product* pipes we have the Cartesian Product of these two input *RangeBlocks*. Such a Cartesian Product may efficiently be represented with a combination of *RLEBlocks* for *Dimension* t and *RangeBlocks* for *Dimension* v. Next, three *Project* pipes generate three *Extensional Mappings* with identical *Domain* and again just reference positions in their codomains. Notice that up to this point, actual data values have not been read from storage yet. A *Constant* pipe generates another *Extensional Mapping* that yields a constant value "Green" for each combination of GPS.Time and Vessel.Id. A *Extensional Mapping* pipe retrieves elements of Vessel.Location using as input the reference positions produced by the *Project* pipes. Given that GPS.Time and Vessel.Id form in that order the *Domain* of Vessel.Location, then position references do not have to be materialized to data in order to access Vessel.Location. Next, an *Intensional Mapping* pipe evaluates "SeaIce" for each combination of GPS.Time and Vessel.Location. Notice that now, GPS.Time reference positions have to be materialized to TimeInstant values in order to be able to evaluate "SeaIce". Next, predicate "<>" is evaluated between the result of "SeaIce" and the *Constant* "Green". At this stage, the result *Extensional Mapping* yields a boolean value for each combination of reference positions (t,v) from GPS.Time and Vessel.Id. Domain pipe *Select* uses such a boolean *Extensional Mapping* to restrict the combinations to only those that are valid. Next, again a *Project* pipe is applied and finally an *Aggregate* pipe yields the expected result *Extensional Mapping* by grouping by *Dimension* t and counting valid reference positions of v.

It is noticed that only the required data elements from *Dimensions* and *Extensional Mappings* are read from storage when they are required for some evaluation. Actually, in the example plan only values of GPS.Time *Dimension* and Vessel.Location *Extensional Mapping* are obtained from disk. This approach of delaying data access operations as much as possible is called *Late Materialization* and it is a well known technique in column-oriented implementations. It is also important to note that although the same *Extensional Mapping* or *Dimension* might appear various times in a evaluation plan, the appropriate use of buffers by the *Data Storage Manager* should avoid having to read their values from tertiary storage more than once. Specific structures might be required to achieve this as it has already been reported in [39].

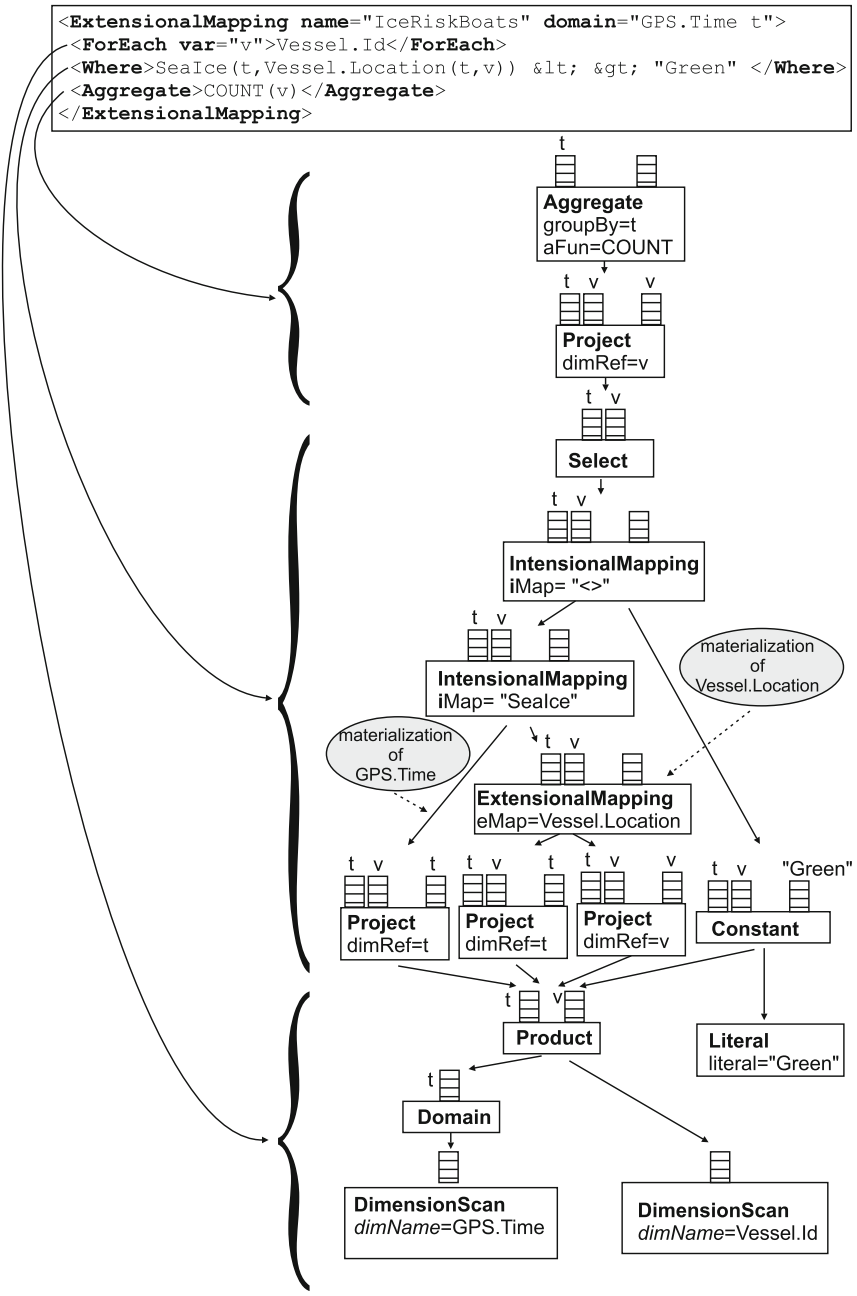

Fig. 8 Example of query evaluation plan

6 Conclusions and further work

A framework for the analysis of spatial observation data was designed and qualitatively evaluated and compared with related data management technologies and approaches. In particular, an observation data model was formalized based on the previous definition of a spatial data model. A declarative spatio-temporal data analysis language was also described and physical implementation issues were discussed. The advantages of the proposed framework may be summarized as follows.

- Observation data semantics are directly supported by the spatial observation data model and also incorporated in the declarative definition of *Internal Processes*.
- The framework provides direct support for the integrated representation and analysis of both conventional E/R data and temporal, spatial and spatio-temporal sampled data.
- The formalized temporal and spatial data types support the representation and transformation between different data resolutions.
- Both data (*Extensional Mappings*) and behavior (*Intensional Mappings*) are represented with the well known mathematical concept of function. Therefore, it is likely that the approach will be friendly to scientific users. Besides, a functional approach simplifies the definition and reuse of intermediate results, enabling the well known software engineering Black Box concept.
- The use of XML syntax of the proposed languages (XODDL and MAPAL) simplifies their incorporation in web service interfaces and enables the use of widely adopted XML technologies in the implementation.
- It is estimated that the use of a single non-nested data structure in the data model will simplify the efficient implementation of the framework.

The main drawback of SODA arise from the assumption of a brand new data management paradigm departing from the classical relational-SQL one, which is a clear handicap for current DBMS users. However, this is alleviated by the fact that well known logical and functional formalisms were combined to define MAPAL, which makes its constructors very similar to those of currently available languages like XQuery.

Future work issues include the following. Complete a first prototype of the framework in a single node multi-core hardware architecture and test its performance in comparison with other relevant solutions; Incorporate query optimization techniques and appropriate indexing structures to improve system performance; Redesign and implement a new version for a multi-node share nothing architecture, exploiting the parallelism with appropriate horizontal partitioning of *Dimensions* and *Extensional Mappings*.

Acknowledgments This work has been partially supported by the Spanish Ministry of Science and Innovation (TIN2010-21246-C02-02). The authors are also grateful to the reviewers, whose comments contributed to greatly improve the paper.

References

1. Cox, S.: Geographic Information—Observations and Measurements. Open Geospatial Consortium (OGC) Abstract Specification Topic 20 and ISO 19156:2011(E) (2013). http://www.opengeospatial.org/standards/om. Accessed Jan 2014
2. Open Geospatial Consortium (OGC): OpenGIS Sensor Model Language (SensorML) Implementation Specification (2007). http://www.opengeospatial.org/standards/sensorml. Accessed Jan 2014
3. Bröring, A., Stasch, C., Echterhoff, J.: OGC Sensor Observation Service Interface Standard. Open Geospatial Consortium (OGC) (2012). http://www.opengeospatial.org/standards/sos. Accessed Jan 2014
4. Bowers, S., Madin, J., Schildhauer, M.: A conceptual modeling framework for expressing observational data semantics. In: Q. Li, S. Spaccapietra, E. Yu, A. Oliv (eds.) Conceptual Modeling - ER 2008, Lecture Notes in Computer Science, vol. 5231, pp. 41–54. Springer, Berlin (2008). doi:10.1007/978-3-540-87877-3_5
5. Compton, M., Barnaghi, P., Bermudez, L., Garca-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Phuoc, D.L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The SSN ontology of the W3C semantic sensor network incubator group. Web Semant. **17**(0), 25–32 (2012). doi:10.1016/j.websem.2012.05.003
6. Madin, J., Bowers, S., Schildhauer, M., Krivov, S., Pennington, D., Villa, F.: An ontology for describing and synthesizing ecological observation data. Ecol. Inf. **2**(3), 279–296 (2007). Meta-information systems and ontologies. In: A Special Feature from the 5th International Conference on Ecological Informatics ISEI5, Santa Barbara, CA, Dec. 4–7, 2006—Novel Concepts of Ecological Data Management S.I. doi:10.1016/j.ecoinf.2007.05.004
7. Neteler, M., Mitasova, H.: Open Source GIS: A GRASS GIS Approach, 3rd edn. Springer, New York (2008)
8. Galpin, I., Brenninkmeijer, C., Gray, A., Jabeen, F., Fernandes, A., Paton, N.: Snee: a query processor for wireless sensor networks. Distrib. Parallel Databases **29**(1–2), 31–85 (2011). doi:10.1007/s10619-010-7074-3
9. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. **30**(1), 122–173 (2005). doi:10.1145/1061318.1061322
10. Güting, R.H.: Spatial Databases. John Wiley, Hoboken (2001). doi:10.1002/047134608X.W4317
11. Lorentzos, N.A., Viqueira, J.R.R.: Relational formalism for the management of spatial data. Comput. J. **49**(1), 62–81 (2006). doi:10.1093/comjnl/bxh136
12. International Organization for Standardization (ISO): Information technology—Database languages—SQL multimedia and application packages—Part 3: Spatial. ISO/IEC 13249–3:2011 (2011)
13. Obe, R., Hsu, L.: PostGIS in Action. Manning, Stamford, CT (2011)
14. Mongodb: http://www.mongodb.org/ (2014). Accessed Jan 2014
15. Idreos, S., Groffen, F.E., Nes, N.J., Manegold, S., Mullender, K.S., Kersten, M.L.: MonetDB: Two decades of research in column-oriented database architectures. IEEE Data Eng. Bull. **35**(1), 40–45 (2012). http://oai.cwi.nl/oai/asset/19929/19929B.pdf. Accessed Jan 2014
16. Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., Widmann, N.: The multidimensional database system rasdaman. In: Proceedings of the 1998 ACM SIGMOD International Conference on Management of data, SIGMOD '98, pp. 575–577. ACM, New York, NY (1998). doi:10.1145/276304.276386
17. Brown, P.G.: Overview of scidb: large scale array storage, processing and analysis. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, SIGMOD '10, pp. 963–968. ACM, New York, NY (2010). doi:10.1145/1807167.1807271
18. Zhang, Y., Kersten, M.L., Manegold, S.: SciQL: array data processing inside an RDBMS. In: Proceedings of ACM SIGMOD International Conference on Management of Data 2013, pp. 1049–1052. ACM, New York, NY (2013). http://oai.cwi.nl/oai/asset/21401/21401A.pdf. Accessed Jan 2014
19. Cugola, G., Margara, A.: Processing flows of information: from data stream to complex event processing. ACM Comput. Surv. **44**(3), 15:1–15:62 (2012). doi:10.1145/2187671.2187677
20. Arasu, A., Babu, S., Widom, J.: The cql continuous query language: semantic foundations and query execution. VLDB J. **15**(2), 121–142 (2006). doi:10.1007/s00778-004-0147-z

21. Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Widom, J., Balakrishnan, H., Çetintemel, U., Cherniack, M., Tibbetts, R., Zdonik, S.: Towards a streaming sql standard. Proc. VLDB Endow. **1**(2), 1379–1390 (2008). http://dl.acm.org/citation.cfm?id=1454159.1454179. Accessed Jan 2014
22. Apache cassandra: http://cassandra.apache.org/ (2014). Accessed Jan 2014
23. Voltdb: http://voltdb.com/ (2014). Accessed Jan 2014
24. Vertica: http://www.vertica.com/ (2014). Accessed Jan 2014
25. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., Zdonik, S.: C-store: a column-oriented dbms. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB '05, pp. 553–564. VLDB Endowment (2005). http://dl.acm.org/citation.cfm?id=1083592.1083658. Accessed Jan 2014
26. Schut, P.: OpenGIS Web Processing Service. Open Geospatial Consortium (OGC) (2007). http://www.opengeospatial.org/standards/wps. Accessed Jan 2014
27. Cerveira Cordeiro, JaP, Câmara, G., Moura De Freitas, U., Almeida, F.: Yet another map algebra. Geoinformatica **13**(2), 183–202 (2009). doi:10.1007/s10707-008-0045-4
28. Date, C.J., Darwen, H., Darwen, H.: Temporal Data and the Relational Model: A Detailed Investigation into the Application of Interval and Relation Theory to the Problem of Temporal. Kaufmann series in data management systems, 1st edn. Morgan Kaufmann Publishers, Inc., San Francisco, CA (2002)
29. Snodgrass, R.T. (ed.): The TSQL2 Temporal Query Language. Kluwer, Philip Drive Norwell, MA (1995)
30. Kulkarni, K., Michels, J.E.: Temporal features in SQL:2011. SIGMOD Rec. **41**(3), 34–43 (2012). doi:10.1145/2380776.2380786
31. Vaisman, A., Zimányi, E.: A multidimensional model representing continuous fields in spatial data warehouses. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS'09, pp. 168–177. ACM, New York, NY (2009). doi:10.1145/1653771.1653797
32. Güting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., Vazirgiannis, M.: A foundation for representing and querying moving objects. ACM Trans. Database Syst. **25**(1), 1–42 (2000). doi:10.1145/352958.352963
33. Viqueira, J., Lorentzos, N.: Sql extension for spatio-temporal data. VLDB J. **16**(2), 179–200 (2007)
34. Baumann, P., Holsten, S.: A comparative analysis of array models for databases. In: Kim, Th, Adeli, H., Cuzzocrea, A., Arslan, T., Zhang, Y., Ma, J., Chung, Ki, Mariyam, S., Song, X. (eds.) Database Theory and Application, Bio-Science and Bio-Technology, Communications in Computer and Information Science, pp. 80–89. Springer, Berlin (2011). doi:10.1007/978-3-642-27157-1_9
35. Gray, P.M.D.: The Functional Approach to Data Management: : Modeling, Analyzing, and Integrating Heterogeneous Data. Springer, Berlin (2004)
36. Sagan, H.: Space-Filling Curves. Springer, Berlin (1994)
37. Abadi, D., Madden, S., Ferreira, M.: Integrating compression and execution in column-oriented database systems. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD '06, pp. 671–682. ACM, New York, NY (2006). doi:10.1145/1142473.1142548
38. Harizopoulos, S., Shkapenyuk, V., Ailamaki, A.: Qpipe: A simultaneously pipelined relational query engine. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD '05, pp. 383–394. ACM, New York, NY (2005). doi:10.1145/1066157.1066201
39. Abadi, D., Myers, D., DeWitt, D., Madden, S.: Materialization strategies in a column-oriented dbms. In: Proceedings of the IEEE 23rd International Conference on Data Engineering, ICDE 2007, pp. 466–475 (2007). doi:10.1109/ICDE.2007.367892