# Towards Progressively Querying and Mining Movement Data

R. Ortale*, E. Ritacco*, N. Pelekis†, R. Trasarti•
G. Costa*, F. Giannotti•, G. Manco*, C. Renso•, Y. Theodoridis†
*ICAR-CNR, Rende (CS), Italy
†University of Piraeus, Piraeus, Greece
•ISTI-CNR, Pisa, Italy

## ABSTRACT

We propose a research foundation for progressively mining and querying both movement data and patterns. Our proposal is based on an algebraic framework, referred to as 2W Model, that defines the knowledge discovery process as a progressive combination of mining and querying operators. The 2W Model framework provides the underlying procedural semantics for a language called MO-DMQL, that allows to progressively refine mining objectives. MO-DMQL extends conventional SQL in two respects, namely a pattern definition mechanism and the capability to uniformly manipulate both raw data and unveiled patterns. Also, an innovative computational engine, DAEDALUS, is introduced for processing MO-DMQL statements. The expressiveness and usefulness of the MO-DMQL language as well as the computational capabilities of DAEDALUS are qualitatively evaluated by means of a case study.

## 1. INTRODUCTION

Research on moving-object data analysis has been recently fostered by the widespread diffusion of new techniques and systems for monitoring, collecting and storing location-aware data, generated by a wealth of technological infrastructures, such as GPS positioning, sensor- and mobile-device networks, tracking facilities. These have made available massive repositories of spatio-temporal data, that call for suitable analytical methods, capable of enabling the development of innovative, location-aware applications.

A flurry of research has covered with spatio-temporal data analysis from different perspectives. The integration of trajectory data with semantic information for more effective analysis was the subject of studies such as [26, 2]. Several approaches to pattern discovery in movement data have emerged [5, 19, 22]. Schemes for approximated trajectory similarity-search, based on nearest-neighbor and indexing schemes, have been proposed in [1, 4]. In a nutshell, so far, research efforts have been largely geared towards either the definition of new movement patterns, or the development of solutions to algorithmic issues, with which to improve existing pattern-mining schemes in terms of effectiveness and/or efficiency. As a consequence, several intelligent tools for movement data analysis have rapidly flourished. In the meanwhile, however, the necessary attention has not been paid to the definition of a unifying framework, wherein to set the above pattern-mining tools as specific components of the knowledge discovery process. Despite some preliminary ideas, it is still an open issue without a predominant proposal. We believe that this is a primary limitation for the real-world applications of movement data analysis, where it rarely happens that a single pattern-mining activity (i.e. either of the foresaid tools) suffices to meet the underlying analytical requirements. Rather, it is often necessary that mining tasks are performed, the resulting models treated on a par with raw data [15, 3] and possibly used for further analysis. Nevertheless, in the current state of the art, the lack of support to knowledge discovery as an actual multi-step process makes impractical all those applications, that involve multiple stages of analysis and manipulation for both data and patterns, in which the results at each stage are required to become the input to the subsequent stage [3]. As a motivating example, consider the following analytical requirement, that calls for the search of common behavioral patterns in the context of specific spatial patterns: *among the routes of objects through a sequence of regions on the left of the city centre, find those common sub-trajectories that traverse a polluted area during rush hours*. Satisfying such a requirement involves a complex two-steps process, wherein multiple mining methods as well as forms of knowledge need be progressively and seamlessly exploited. As a matter of fact, temporally annotated sequences of regions [5] must be initially uncovered in primary knowledge (i.e. the raw trajectory data). Next, common sub-trajectories [22] have to be mined from the sequence sited on the left of the city centre and eventually related with background knowledge (i.e. polluted areas and rush hours) to distill the desired patterns. Unfortunately, in the absence of a unifying framework, the process of progressively querying and mining both movement data and patterns is in general a challenging issue. Indeed, the individual mining techniques can hardly be combined into an actual multi-step process, since their results are typically neither directly exploitable to feed some further analysis, nor uniformly manageable with raw data.

Interestingly, some aspects of the problem have been partly touched by various proposals in the field of moving-object databases [6]. The latter have mainly focused on the development of comprehensive algebraic frameworks for modeling and progressively querying (un)constrained movement data [13, 14]. The underlying intuition essentially consists in extending the relational model, which provides the basic foundation for progressive querying, with a set of abstract data types (representing generic moving entities), equipped with related spatio-temporal primitives. Unfortunately, the resulting frameworks are not meant for knowledge discovery purposes.

In the present paper, to the best of our knowledge, we take a first step towards progressively mining and querying movement data. In this direction, our contribution is threefold. We introduce MO-DMQL, a specific language capable of supporting the user in specifying and refining mining objectives. The procedural semantics of the language is founded on an algebraic framework, referred to as the 2W Model, which allows to progressively accommodate mining and querying tasks, over both movement data and patterns, into a multi-step knowledge discovery process. The MO-DMQL language and its semantics are embedded into an innovative computational environment, called DAEDALUS, that provides effective support to the whole knowledge discovery process, by transparently translating MO-DMQL statements issued from the end user into executions of specific mining/querying tasks.

The rest of the paper proceeds as follows. Section 2 provides a survey of approaches available from the current literature, that have posed the basis of our proposal. Section 3 gives the formal definition of the algebraic framework 2W Model. Section 4 proposes, MO-DMQL, a data-mining query language for progressively querying and mining movement data, whose procedural semantics is founded on the 2W Model. Section 5 discusses the design of the DAEDALUS system for processing MO-DMQL queries. Section 6 qualitatively evaluates the expressiveness and usefulness of the MO-DMQL language as well as the computational capabilities of DAEDALUS by means of a case study. Finally, section 7 concludes the paper by drawing some conclusions and highlighting major directions of further research.

## 2. RELATED WORK

We review some influential contributions in the current literature from three different perspectives: querying moving objects, mining movement patterns and progressive mining and querying.

### 2.1 Querying Moving Objects

Research on moving-object databases has addressed the need for representing movements of objects (i.e. trajectories) in databases, in order to perform ad-hoc querying, analysis, as well as knowledge extraction. In particular, several research initiatives in the last decade focused on data models and query languages for moving-object databases. We can devise two main research directions: the first focuses on querying current and future positions of the moving objects in [17, 11, 16], while the second on querying past trajectories of the moving objects in [13, 7, 12].

The first approach by Wolfson et al. in [11, 16] is challenged by the issue of how often location updates should be sent and applied to the database, to face a trade-off between error in location information and update load. The authors propose the so-called Moving Objects Spatio-Temporal (MOST) data model for databases with dynamic attributes, i.e. attributes that change continuously as a function of time, without being explicitly updated. The query language FTL (Future Temporal Logic) based on temporal logic is introduced to formulate questions about the near future movement.

The second approach is concerned with the study of abstract moving-object data types, as well as algorithms supporting the related operations [13]. Forlizzi et al. [12, 7] provide a systematic study of a fragment of the methods introduced in [13], and offer a blueprint for implementing a database extension package tailored towards moving objects. The final outcome of this work has been recently demonstrated in [20]. The Hermes framework, has been recently introduced by Pelekis et al. [25], that aims at aiding a database developer in modeling, constructing and querying a moving-object database. Hermes is a basic component of the proposed the DAEDALUS framework, introduced in section 5.

### 2.2 Mining Movement Patterns

There has been considerable work on the development of methods for knowledge discovery from movement data. Proposals comprise approach for unveiling both global and local movement patterns. We here review some influential techniques. The interested reader is referred to a more comprehensive survey in [8].

Clustering has been naturally exploited to uncover a variety of global behavioral patterns. The problem of discovering dense regions is covered in [24]. Here, the goal is twofold, i.e. both discovering those areas than are deemed to contain more than a certain threshold of objects in a specified time period and finding for how long those areas can still be considered dense.

A generative probabilistic approach is used in [21] to model the individual trajectories as sequences of points yielded by a finite mixture of regression models. Clusters of such sequences are then found by means of a suitable EM estimation procedure.

The approach in [19] is devoted to the discovery of moving clusters, i.e. groups of objects that move similarly and close to each other for a long time. Differently from the methods in in [24, 19], resulting in cluster that are, respectively, either static in space or in content over time, moving clusters do not necessarily maintain their inner content: moving objects can join and/or leave any moving clusters at any given point in time. The only requirement is that the individual moving cluster preserves its density over its lifetime.

The identification of local patterns in movement data, i.e. of concise representations of interesting local behavioral patterns of moving objects, has been also a fertile area of research. The approach in [22] is devoted to the identification of common sub-trajectories. This is accomplished by partitioning the individual routes into line segments by means of the minimum description length principle and then grouping such segments through a density-based clustering strategy. Each resulting cluster is equipped with a representative common sub-trajectory, summarizing the overall movement of the line-segments in the spatial region associated to the group. Trajectory pattern mining in [5] is instead pursued to unveil T-patterns, i.e. sequences of temporally-annotated spatial regions. Each T-pattern concisely represents all those trajectories of moving points that share the common property of visiting the same chronologically-ordered sequence of places (i.e. spatial regions) with nearly similar travel times. The T-pattern mining scheme is currently implemented in the mining engine of the DAEDALUS system, discussed in section 5.

### 2.3 Progressive Mining and Querying

The last decade has seen a proliferation of approaches to Data Mining query languages, proposed with different focuses. In a first research direction, the focus is to provide an interface between data sources and data mining tasks. Under this perspective, a DMQL is seen as a standard mean for specifying data sources, patterns of interest and properties characterizing them. In a second direction, a DMQL is meant to support the design of specific procedural workflows which integrate reasoning on the mining results and possibly define ad-hoc evaluation strategies and activations of the Data Mining tasks. Therefore, the idea here is to embody Data Mining query languages in a more general framework, where effective support to the whole knowledge discovery process is provided.

A recent survey on the above mentioned issues appeared in [9]. To the purpose of this paper, however, it is important to briefly review the approach proposed in [18] and subsequently refined by [3], namely, the 3W Model. 3W Model stands for *Three Worlds for data mining*: the D(ata)-world, the I(ntensional)-world, and the E(xtensional)-world. The D-World represents the raw data to be analyzed in terms of the basic entities of relational algebra, i.e. relational schemas and extensions. The attributes of such entities are associated with corresponding domains, that can be either categorical or numeric. Most activities, carried out in the preprocessing phase of a typical knowledge discovery application, can be modeled by means of specific operators of an extended relational algebra, that adds to the usual algebraic operators.

Objects in the I-World represent, instead, a particular class of data mining models, i.e. regions that can be defined as (sets of) conjunctions of linear inequality constraints on the attributes of the entities in the D-World. Starting from a set of basic regions, further regions can be formed via the definition of composition operators.

In the E-World, a region is simply represented as an enumeration of all the tuples belonging to that region. Relations in this world are obtained by combining the relations of the two worlds previously defined, so that the schema of the resulting relation is the union of the schemas of some relation in the D-World and some other relation in the I-World. Thus, the resulting 3W Model can be specified as a set of three worlds: the D-World (data world), the I-World (intensional world), and the E-World (extensional world). Entities in the three foresaid worlds can be related via suitable inter-world operators. Precisely, a generic mining operator *regionize* extracts regions in the I-World from data in the D-World. These regions can be iteratively refined by means of a *mining loop* from the I-World to the I-World. The populate operator *POP* creates a relation in the E-World, starting from some regions in the I-World and some other relations in the D-World. Finally, composite objects of the E-World can be projected to the other two worlds via the operators $\pi_{RDA}$ and $\pi_A$, that allow to return in the I-World and D-World, respectively, via a simple selection of the proper attributes (data or constraints) within the E-World relation.

The 3W Model is mightily interesting for many reasons. Foremost, it provides a view of data mining in algebraic terms: a knowledge discovery process is the application of a sequence of operators in order to transform a set of tables. Furthermore, it is also fascinating from a methodological point of view: the object representation of 3W Model entities and the implementation of a suitable set of operators are key elements in the design of a powerful tool for knowledge discovery. However, some major limitations affect the 3W Model. In the D-World there is no possibility to express complex relations (i.e. cyclic relation), because the nesting of this data model has a fixed depth. Furthermore, a more serious limitation lies in the I-World, where regions are expressed by linear inequality sets. This means that fundamental mining models are not expressible, since their representations require more complex mathematic structures (i.e. SVM and clustering results, time point series, surrounding regions and so forth). The 2W Model in section 3 avoids both the foresaid limitations of the 3W Model. Indeed, it enables the description of complex objects and their properties and also supports the extraction all required patterns from raw data.

## 3. THE 2W MODEL FRAMEWORK

The proposed 2W Model summarizes the essence of a knowledge discovery process, within any applicative setting, as the interaction between two (apparently) different worlds: the *data* world and the *model* world. Whenever each world is populated by the appropriate entities, a set of operators can be defined and used, in order to specify any complex process targeted at the extraction of actionable knowledge.

Within the 2W Model, we explicitly model a knowledge discovery process as a functional expression relating entities in the two worlds, as shown in fig. 1. There are three main kinds of interaction between data and models:

- Filtering functions are self-injecting operations: indeed, they take a set of entities as input and produce a new set of entities. Within the figure, the *Data Filtering* and *Model Filtering* arrows denote such operations.
- Mining functions relate data entities to model entities. In practice, such operations correspond to the application of a data mining algorithm to a given data source. The result is a composite object, describing a pattern holding over such data sources.
- Apply functions are meant to model a sort of dual operation w.r.t. mining functions. In general, a model is a specification of a set of properties holding in the data. Applying a model to a data source essentially means making such properties explicit in extensional form: e.g., by associating each trajectory in a table with the most likely target class according to the model, or by enumerating the frequent patterns appearing within the trajectory.

For the definition of the contours of the two worlds and their operators, one has to concentrate on which entities (i.e. which patterns) are supported in the model world, how data entities relate to model entities, and how constraint solving takes place. The formalization of such aspects strictly depends on the nature of the underlying applicative domain and pursued objectives. Subsections 3.1 and 3.2 provide, respectively, an instantiation the D-World and M-World in the context of movement data analysis.

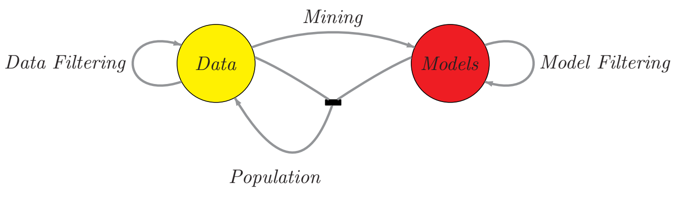

**Figure 1: The 2W Model**

### 3.1 The D-World

The D-World represents the entities to be analyzed, as well as their properties and mutual relationships. Raw data is organized in an object-relational format. The D-World can be viewed as a database $\mathcal{D} = \{r_1(\mathcal{R}_1), \ldots, r_n(\mathcal{R}_n)\}$ of meaningful entities. The generic entity $r(\mathcal{R})$ is a relation with schema $\mathcal{R}$. Formally, $\mathcal{R} = \{A_1 : \texttt{Dom}(A_1), \ldots, A_h : \texttt{Dom}(A_h)\}$, where $A_1, \ldots, A_h$ correspond to descriptive attributes of the data within $r(\mathcal{R})$ and $\texttt{Dom}(A_1), \ldots, \texttt{Dom}(A_h)$ are their respective domains. Relation $r(\mathcal{R})$ is defined as $r(\mathcal{R}) \subseteq \texttt{Dom}(A_1) \times \ldots \texttt{Dom}(A_h)$. Attribute domains can be either primitive or object data types. Primitive types are assigned to simple features of the data and divide into categorical and numerical domains. Instead, object data types abstractly represent complex real-world entities as objects, equipped with application-dependant operations. Hereafter, we omit the specification of relation schema and use the resulting simplified notation to indicate an entity of $\mathcal{D}$. Furthermore, we denote by $t \in r$ a tuple of relation $r$ and, also, exploit notation $t[A_i]$ to indicate the value of tuple $t$ over a schema attribute $A_i$. So far, the description of the D-World is general enough to be employed within any applicative setting. Since we aim at dealing with movement data, hereafter the D-World is assumed to be a repository of movement data. From this point of view, relation schemas involve object data types, modeling the addressed moving entities (such as points and regions).

To elucidate, we introduce the reference relation `Trajectories`, that shall be used throughout the paper to describe pedestrian and/or vehicle routes. Depending on the specific modeling requirements, a possible choice of its schema attributes may comprise `ID` of type integer, `Type` which takes on the categorical values (i.e. `vehicle` and `pedestrian`), and `Trajectory` that is of an object data type, named `Moving_Point`. The latter object type actually models the notion of trajectory. More precisely, given a tuple $t \in$ `Trajectories`, $t$[`Trajectory`] represents a sequence of object locations evolving over time and, also, provides a set of basic operations, for manipulating route data as well as answering topological and distance queries. An in-depth coverage of the `Moving_Point` object type is beyond the scope of the paper. See [25] for further details on `Moving_Point` as well as other object types, that can in principle be used as basic domains in the D-World to represent a wealth of distinct moving entities.

**D-World Operators.** Data in the D-World is manipulated via the usual (unary and binary) operators of traditional relational algebra, namely $\rho, \sigma, \pi, \cup, \cap, \setminus$ and $\times$. Also, aggregation functions (such as `SUM`, `COUNT`, `AVERAGE`, `MIN` and `MAX`) are allowed to operate on collections of domain elements. Algebraic operators can be used to model suitable data preparation/manipulation tasks. As an example, the composite operator below

$$\pi_{\texttt{Trajectory}}(\sigma_{\texttt{Type = "vehicle"}}(\texttt{Trajectories}))$$

represents a trivial reduction of data size and dimensionality.

More complex preparation/manipulation tasks can be expressed by incorporating the basic operation of the (domain-specific) object-relational entities in the corresponding algebraic formulation. To exemplify, a basic operation of the `Moving_Point` data type is `intersects`, which queries whether two trajectories encounter each other. Such a functionality can be exploited to filter from `Trajectories` and count all those vehicle routes that encounter, somewhere and at any given point in time, the route followed by a reference moving point (i.e. with a specified identifier). To this purpose, by means of the expression

$$T = \rho_{Route \leftarrow Trajectory}(\sigma_{\texttt{ID = 3}}(\texttt{Trajectories}))$$

one obtains a new answer relation `T` consisting of the route followed by the moving point with `ID=3`. Here, for convenience, the `Trajectory` attribute of `T` is renamed as `Route`.

As to the aggregation functions, these are not relational algebra operators. In principle, they are used as parameters of some suitable aggregate formation operator. In our formalization, we express queries involving aggregates by means of suitably extended projection operators, in the spirit of the idea in [23], that allow the incorporation of aggregation functions into the algebra. For instance, consider to join `T` with `Trajectories` to count those routes that intersect the one in `T`. This query can be expressed as

$$\pi_{\texttt{count(Trajectory)}}(\sigma_{Trajectory.intersects(Route)}(Trajectories \times T))$$

where $\pi_{\texttt{count(Trajectory)}}(\cdot)$ returns the size of the column `Trajectory` if it appears in the input relation, 0 otherwise. Extended projection operators come in two flavors, depending on whether relation columns are viewed as bags.

Notably, spatio-temporal operations allow the qualitative formulation of queries over movement data, which is relevant for two major reasons. Firstly, it permits to abstract away from the huge amount of punctual data, inherent in the complexity of trajectory data. From this perspective, query formulation can be considered as closer to the way humans reason. Secondly, it somehow allows to deal with imprecision and uncertainty.

Finally, as it shall be clear from the elaboration in subsection 3.2, the D-World algebraic operators also play a fundamental role in post-processing the data resulting from the application of a pattern in the M-World to raw route data.

## 3.2 The M-World

Movement patterns concerning data entities, their properties and relationships are mapped to objects in the model world, which provides an elegant and expressive framework for both exploratory analysis and reasoning. Different types of movement patterns can be defined to populate the M-World, on the basis of the specific applicative requirements.

Formally, the M-World can be represented as a collection of pattern objects $\mathcal{P} = \{p_1, \ldots, p_v\}$, unveiled at the different stages of the knowledge discovery process. The latter are instances of various abstract data types $P_1, \ldots, P_t$ (with $t \leq v$). The class of all instances of the generic abstract data type $P$ is defined as $I(P) = \{p \in \mathcal{P} | p : P\}$, where ":" indicates instantiation. Inside $I(P)$, instances are univocally distinguished by means of a unique identifier. Depending on the nature of $P$, $I(P)$ consists of either singleton or composite pattern objects. The former are individual models such as classifiers. The latter correspond to suitable collections of singleton pattern objects. From an expressiveness point of view, collections are relevant as they allow to capture the natural behavior of several mining tasks: for instance, a clustering task yields a collection of clusters, whereas the outcome of a pattern mining algorithm is usually represented by a set of singleton patterns. Structurally, any composite pattern object $p$ is modeled as an aggregation and equipped with a *set* attribute, that comprises the singletons grouped by $p$. In the rest of the paper, for readability sake, we shall denote the $i$-th singleton pattern in $p$ by $p[\![i]\!]$ rather than using notation $p.set[\![i]\!]$. Also, we shall interchangeably use the notions of composite pattern object and object collection.

Abstract data types also declare the properties that can be carried out on their instances. All pattern objects within each class $I(P)$ expose a common interface $\mathcal{F}_P$, that can be formalized as a schema $\mathcal{F}_P = \{f_1, \ldots, f_m\}$ (with $m > 0$), where individual features are class operations. In our formalization, each $\texttt{f} \in \mathcal{F}_P$ is an instance operation, i.e. an operation accessible trough a host pattern object $p \in I(P)$, which is hereafter indicated as $p.\texttt{f}$. Clearly, if $P$ models a class of composite pattern objects, it also holds that $[\![]\!] \in \mathcal{F}_P$. Notwithstanding, as already anticipated, we use the more readable notation $p[\![\ldots]\!]$ to mean the exploitation of the subscript operator.

The generic operation $\texttt{f} \in \mathcal{F}_p$ can be defined into two alternative ways, depending on the type of its input parameters. Precisely, assume that `f` requires data from the tuples of a D-World relation $r$ with schema $\mathcal{R} = \{A_1, \ldots, A_h\}$. In particular, let $1 \leq i_1 \leq \ldots \leq i_k \leq h$ represent the coordinates of the values in each tuple $t \in r$ in input to `f`. In such a case, the latter operation can be defined as $\texttt{f} : I(P) \times \mathcal{V}[\times Dom_1 \times \ldots \times Dom_u] \rightarrow \mathcal{A}$, where $I(P)$ is the class of host objects on which `f` can be invoked and $\mathcal{V} \equiv Dom(A_{i_1}) \times \ldots \times Dom(A_{i_k})$. Also, $\mathcal{A}$ coincides with $\mathcal{P}$ if `f` returns a pattern object; it is otherwise any categorical or numeric set. Yet, $Dom_1, \ldots, Dom_u$ represent the domains of some additional input arguments that, in principle, may be required by $f$. The foregoing definition remains almost unchanged whenever `f` is fed with pattern objects from the M-World, apart from domain $\mathcal{V}$

that, in such cases, is replaced by $\mathcal{P}$.

Figure 2 exemplifies the notions of composite and single abstract data types. The illustration shows the definition of the *T-pattern* abstract data type, which intuitively represents a sequence of regions traversed by some moving entities within certain time constraints [5]. We here use a pseudo F-Logic [10] formalism, where a type is associated with a type name (in bold) and consists of a collection of properties, enclosed by square brackets. For each property, the double-shafted arrow $\Rightarrow$ stands for type declaration, whereas notation @ divides property name (on its left) from the formal-parameter types (on its right). Notably, the composite abstract data type `T_Patterns_ADT` is formalized as a set of singleton `T_Pattern_ADT`s, individually accessible through the operator $[\![\,]\!]$. In turn, `T_Pattern_ADT` is modeled as an annotated sequence of two-dimensional spatial regions. The foresaid annotations are stored in the `transition_intervals` attribute, a set of time intervals. The sequence of traversed regions is instead hold by `region_sequence`. The $i$-th interval within `transition_intervals` catches the minimum and maximum amount of time required for the underlying trajectories to traverse the corresponding $i$-th region in `region_sequence`.

```
T_Patterns_ADT[
   set ⇒ SetOf(T_Pattern_ADT);
   operator ⟦⟧@int ⇒T_Pattern_ADT;
   west@Region ⇒ T_Patterns_ADT;
   east@Region ⇒ T_Patterns_ADT;
   south@Region ⇒ T_Patterns_ADT;
   north@Region ⇒ T_Patterns_ADT;
]

T_Pattern_ADT[
   transition_intervals ⇒ SetOf{Time_Interval};
   region_sequence ⇒ SetOf{Region};
   contains@Moving_Point ⇒ boolean;
   crossed@Moving_Point ⇒ boolean;
   disjoint@Moving_Point ⇒ boolean;
   lies@Region ⇒ boolean;
   overlaps@Region ⇒ boolean;
]
```

**Figure 2: The T_Patterns_ADT abstract data type.**

In principle, different abstract data types provide distinct features. Furthermore, there exists a wealth of applicative scenarios, which require various modeling properties for the same abstract data type. An exhaustive analysis of all possible abstract data types in the different applicative scenarios is clearly infeasible. Therefore, we next provide an insight into class operations as well as their possible uses in practice, by focusing on the abstract data types of fig. 2 and briefly discussing the operations reported in the illustration.

Let us suppose that $p$ : `T_Patterns_ADT` is a composite pattern object extracted from the foregoing `Trajectories` relation. Several useful operations can be envisaged for the generic singleton $p[\![i]\!]$ : `T_Pattern_ADT`, whose definition is however omitted due to space restrictions. In particular, with respect to a tuple $t \in$ `Trajectories`, the three operations `contains`, `crossed` and `disjoint` can be used, respectively, to ask whether the route of the `Moving_Point` argument falls inside, intersects, does not meet $p$. Interestingly, some operations can also be employed to manipulate data other than `Trajectories`, from which $p$ was originally extracted. For instance, assume that the D-World stores a further relation `Places`, whose schema includes a region identifier `ID` and an object-relational attribute `Region`, modeling two-dimensional spatial boundaries. For any place $t' \in$ `Places`, one can use the spatial features `lies` and `overlaps` to verify whether $p[\![i]\!]$ lies, respectively, partly overlaps the place area delimited by $t'$`[Region]`. Suitable operations can also be defined for the composite object $p$. These are however devoted at retrieving specific singletons. In the spirit of fig. 2, retrieval operations use the positional information `region_sequence` within the individual singletons $p[\![i]\!]$, to find out the ones sited, respectively, to the west, east, south, north of a certain place. Precisely, given a place $t' \in$ `Places`, $p$.`west(t'[Region])` returns a new `T_Patterns_ADT` instance, whose `set` attribute is narrowed to the singletons of $p$ that reside to the west of `t'[Region]`.

Interactions between raw data within the D-World and the objects in the M-World are the basic building-blocks for the definition of multi-step knowledge discovery processes. Interactions correspond to 2W Model operators, that establish connections between two worlds. Operators acting on patterns in the M-World divide into mining and population operators. Precisely, starting from some raw data in the D-World, mining operators define the patterns that populate the M-World. Instead, population operators specify how to create new data in the D-World. Mining and population operators are the subjects of the following paragraphs.

## 3.3 Mining and Population Operators

The population of the M-World starting from the data in the D-World is delegated to the so-called *mining operator* $\kappa$. Let $r_{i_1}, \ldots, r_{i_l}$ be a subset of relations in $\mathcal{D}$ from which to extract some suitable movement pattern. The inter-world mining operator $\kappa$ is defined as $\kappa : \mathcal{D}^l \rightarrow \mathcal{P}$. The resulting object is an instance of one of the available abstract data types $P_1, \ldots, P_t$ mentioned in subsec. 3.2.

To exemplify, if $\kappa$ represents the mining scheme in [5] and `Trajectories` is the reference relation introduced in subsection 3.1, $\kappa$(`Trajectories`) extracts an object $p$ : `T_Patterns_ADT`.

Once accomplished the forward population of the M-World with the required patterns, their operations can be employed in the opposite direction, i.e. to backward populate the D-World with further data. Indeed, pattern operations allow the definition of suitable criteria, required to suitably populate the D-World with new data. Interestingly, this does not involve the explicit representation of further (composite) objects as in the E-World of the 3W Model. Simply, the raw data of a relation $r$ with schema $\mathcal{R}$, that meets some population criterion (i.e. any constraint on a certain pattern operation), originates a new relation $r'$ in the D-World whose schema is still $\mathcal{R}$. The inter-world *population operator* $\ltimes_{Constr(f_i(\cdot))}$: $\mathcal{P} \times \mathcal{D} \rightarrow \mathcal{D}$, is used to formalize the semantics of population process in terms of constraints on pattern operations. Precisely, given a pattern $p \in \mathcal{P}$ and a relation $r \in \mathcal{D}$, notation $Constr(f_i(\cdot))$ represents the constraint on $f_i(\cdot)$, which must be satisfied by all tuples $t \in r$ that ultimately populate the resulting relation $r'$. Notably, the procedural definition of the population process slightly varies depending on the nature of the argument model $p$. More specifically, if $p$ is a singleton pattern object, the outcome of the population operator $\ltimes_{Constr(f_i(\cdot))}$ is simply the subset of all tuples in $r$ that satisfy the corresponding constraint on $p.f_i(\cdot)$:

$$r' = p \ltimes_{Constr(f_i)} r = \{t \in r | Constr(p.f_i(t)) \;\; holds\}$$

Instead, if $p$ is a composite pattern object, the population operator identifies the union of all tuples in $r$, that satisfy any constraint $Constr(p[\![j]\!].f_i(\cdot))$, for each $p[\![j]\!] \in p$:

$$r' = p[\![\cdot]\!] \ltimes_{Constr(f_i)} r = \{t \in r | p[\![j]\!] \in p, Constr(p[\![j]\!].f_i(t)) \ holds\}$$

To exemplify, assume that $p$ is an instance of T_Patterns_ADT and, hence, groups a number of singleton T_pattern_ADT objects $p[\![1]\!], \ldots, p[\![n]\!]$. The expression

$$\texttt{Trajectories}' = p[\![i]\!] \ltimes_{\texttt{contains}} \texttt{Trajectories}$$

originates a new relation Trajectories′ in the D-World, with a same schema as the one of the original relation Trajectories. Trajectories′ is populated with those moving objects from Trajectories, whose route is inside the individual T-pattern $p[\![i]\!]$. On the contrary, the below expression

$$\texttt{Trajectories}' = p[\![\cdot]\!] \ltimes_{\texttt{contains}} \texttt{Trajectories}$$

populates Trajectories′ with the moving objects of Trajectories, whose routes is inside any singleton in $p$.

## 3.4 Discussion

The 2W Model introduces several meaningful differences w.r.t. the 3W Model. Firstly, entities in the M-World can represent any required patterns, even if with a mathematically complex structure, whereas I-World models correspond to simple regions, expressible via linear inequalities on the data attributes. Secondly, in the 2W Model, $\kappa$ is not predefined and acts as a template to extract a model from a table. Thirdly, we directly map objects in the E-World to counterparts in the D-World, without an explicit representation in the E-World. By definition of population operator, the application of any model to the data in a relation of the D-World always produces a further relation within the D-World. This ensures that mining results can be progressively analyzed on a par with raw data, via further manipulations. To exemplify, consider the case where one wishes to uncover the common sub-trajectories of those moving points, whose route is inside a specified T-pattern. In such a case, T-patterns [5] are first extracted into the M-World via a specific mining operator $\kappa_1$. Data and unveiled patterns are then treated on a par to the purpose of identifying the trajectories inside the $l$-th T-pattern, which is accomplished by applying the contains feature to the trajectory data. Common sub-trajectories [22] are then discovered within the specified T-pattern, by applying a second mining operator $\kappa_2$ to the new raw data. In the 2W Model, the algebraic formulation of the foresaid knowledge discovery process is

$$\kappa_2(\kappa_1(\texttt{Trajectories})[\![l]\!] \ltimes_{\texttt{contains}} \texttt{Trajectories})$$

from which it clearly appears that $\kappa_1$ and $\kappa_2$ are pipelined via the population operator.

Finally, we recall that the D-World operators contribute to the expressiveness of the 2W Model framework, by playing a twofold role. On the one hand, such operators can be used to represent preprocessing tasks, e.g. the reduction in size and/or dimensionality of the available data. On the other hand, they are useful for postprocessing purposes, such as in the act of filtering interesting patterns.

The 2W Model is adopted in section 4 as a foundation for the definition of the procedural semantics of a language, designed for interactively querying and mining movement data.

## 4. IMPLEMENTING THE 2W MODEL

The 2W Model is a natural foundation for the development of domain-specific data mining query languages. Within a specific applicative domain, this mainly involves the definition of appropriate mining and population operators, as well as the specification of the basic object-relational entities. In the present paper, we propose MO-DMQL, a data mining query language, designed to support knowledge discovery from movement data as an actual multi-step knowledge discovery process. The intuition consists in starting from the conventional SQL language, that provides basic mechanisms for interactively querying and manipulating the entities within the D-World (i.e. both original raw data and the outcome of population operators). These are extended in two major respects. Firstly, the introduction of a pattern definition statement, i.e. CREATE MODEL, for the specification of the required movement models, with which to populate the M-World. Secondly, the capability of supporting generic population operators, which ultimately allows the application of models in the M-World to raw data within the D-World. For this reason, we revised the traditional join semantics inherent in the SELECT-FROM-WHERE statement, so that raw data and unveiled patterns can be uniformly manipulated and joined for further analysis. The semantics of the inherent MO-DMQL statements is elaborated next.

## 4.1 Model Definition

CREATE MODEL implements the mining operator $\kappa$ of the 2W Model and builds a particular model in the M-World. Its syntax is reported below.

```
CREATE MODEL <model_name> AS MINE <mining_algorithm>
FROM <<table>>
WHERE <mining_algorithm>.param_1 = val_1 ⊗ ... ⊗
      <mining_algorithm>.param_n = val_n
```

The above statement specifies a pattern-discovery task, via a call to some corresponding mining algorithm. In terms of the 2W Model algebra, the definition creates a model object in the M-World named <model_name>, according to the procedural semantics $k(\texttt{table})$, where the effect of the mining operator $\kappa$ is the application of the <mining_algorithm> to table. In this respect, an important difference with respect to the traditional SQL CREATE statement is that the latter guarantees closure by returning a table, so that further SQL statements can be issued over it. Instead, the CREATE MODEL statement results into a (singleton or composite) object, that is an instance of some corresponding abstract data type. Closure is enforced by the possibility of manipulating both raw data and pattern objects, described in subsection 4.2.

The CREATE MODEL statement enables the development of data-mining query languages, that meet user's requirements in any given applicative setting. In this paper we focus on movement data analysis and, hence, assume that, hereafter, <mining_algorithm> denotes any methods for discovering movement patterns. <<table>> denotes the primary trajectory data of the D-World, from which <model_name> is extracted, that can be in the form of either a materialized database table, a view, or a query. The WHERE clause allows to properly specify the input parameters of the invoked <mining_algorithm>, that involve algorithm-specific parameters, search biases and thresholds for interestingness measures. Combinations of logical conditions on input parameters are expressed via any connectors $\otimes$ from traditional SQL grammar.

Notice that, depending on the <mining_algorithm>, background knowledge within some further table of the D-World can be directly taken into account to assist pattern discovery, by either specifying further tables in the FROM clause or exploiting the relational organization of the trajectory data. This is useful for several reasons, such as either enriching the data at hand, deriving good initial hypotheses with which to start the search for patterns, defining preference biases that prune the pattern search space, or providing a reference for the interpretation of the discovered patterns. Furthermore, since it is often difficult to define adequate statistical measures for subjective concepts like novelty, usefulness, and understandability, background knowledge can be also helpful in capturing such concepts more accurately.

The following `CREATE MODEL` statement exemplifies the definition of a T-pattern mining task, which requires the availability in the D-World of the `Trajectories` table introduced in subsection 3.1.

```
CREATE MODEL T_Patterns AS MINE Dynamic_TPattern_Mining
FROM Trajectories
WHERE Dynamic_TPattern_Mining.density = δ  AND
      Dynamic_TPattern_Mining.snr = ε  AND
      Dynamic_TPattern_Mining.tt = τ
```

The `T_Patterns` composite object is instantiated from the abstract data type `T_Patterns_ADT` of figure 2, by applying the `Dynamic_TPattern_Mining` to the basic `Trajectory` data. Algorithm-specific parameters appear in the above syntax as features of the `T_Patterns` object and are suitably set in the `WHERE` clause. Here, the minimum density threshold (`density`), spatial neighborhood radius (`snr`) and temporal threshold (`tt`) are set to suitable values, respectively represented by $\delta$, $\epsilon$ and $\tau$. Further details on the T-pattern mining algorithm and the mentioned input parameters are provided in [5].

## 4.2 Data and Model Manipulation

The `SELECT-FROM-WHERE` statement can be used in MO-DMQL to accomplish several different tasks of the knowledge discovery process. The procedural semantics of the individual statement is defined as some suitable combinations of 2W Model operators. In the following, we elucidate the `SELECT-FROM-WHERE` statement in the manipulation of raw data as well as in the definition and further analysis of movement patterns.

**Raw Data Manipulation.** Data manipulation and querying represents the simplest exploitation of the statement. Query $\mathbf{Q_1}$ defines a simple preprocessing of trajectory data, before it is used in any subsequent analytical task.

```
SELECT Trajectories.id, Trajectories.Trajectory
FROM Trajectories
WHERE Trajectories.type="vehicle"
```

Clearly, $\mathbf{Q_1}$ filters vehicle trajectories and projects them on attributes `ID` and `Trajectory`, deemed relevant for subsequent pattern discovery. In terms of D-World operators, the procedural semantics of the above statement is

$$\pi_{\texttt{ID,Trajectory}}\left(\sigma_{\texttt{type="vehicle"}}(\texttt{Trajectories})\right)$$

In the remainder of this subsection, we assume that the model definition statement in subsection 4.1 is issued over the answer to $\mathbf{Q_1}$ to specify the discovery of a model named `T_Patterns`.

**Hybrid Manipulation of Raw Data and Models.** The manipulation of pattern objects in the M-World enables more advanced uses of the `SELECT-FROM-WHERE` statement. By suitably joining data within the D-World and models in the M-World, it is possible to find out the raw data that meets a particular constraint on some feature of a certain pattern. This is exemplified next, by means of increasingly expressive formulations of a same query $\mathbf{Q_2}$. We start with the selection of all trajectories inside the $l$-th T-pattern of the foresaid model `T_Patterns`.

```
SELECT T.*
FROM   Trajectories T, T_Patterns TS
WHERE TS⟦l⟧.contains(T.Trajectory)
```

Here, notation `TS`$[\![l]\!]$ denotes the $l$-th pattern of the `T_Patterns` object. The hybrid nature of the entities within the `FROM` clause reveals that the semantics of $\mathbf{Q_2}$ can be defined as

$$\kappa_1(\texttt{Trajectories})[\![l]\!] \ltimes_{\texttt{contains}} \texttt{Trajectories}$$

which corresponds to a specific instantiation of a 2W Model populate operator. The above query is not really user-oriented, since it provides the user with a nonrealistic mechanism for manipulating both data and models, which forces her/him to tentatively find the desired answer to her/his own requirements. In general, a user is not expected to blindly issue such a query, in the absence of further information with which to contextualize the question for the $l$-th pattern. Whenever contextualization is necessary for query formulation, MO-DMQL allows the formulation of more sophisticated statements, in which questions like the $l$-th singleton are functional to identify one pattern in a subset of singletons satisfying a same criterion. As an example, let us assume that the user is interested in the T-patterns sited on the left of the city centre and that, from the visual inspection of the city road map, it is possible to identify a certain number of such patterns in the specific area of the city map. In such a case, the user may properly ask for the trajectories included in the $l$-th of such T-patterns, to better understand and/or further analyze mobility throughout the specific city area. The below statement reformulates $\mathbf{Q_2}$ to meet the foresaid requirement and involves table `Places` (introduced in section 3.2), that here is assumed to provide a localization for the different areas of the city map:

```
SELECT T.*
FROM   Trajectories T, T_Patterns TS, Places P
WHERE Places.id="centre",
      (TS.west(Places.Region))⟦l⟧.contains(T.Trajectory)
```

Procedurally, answering to the above statement involves two steps, i.e. singleton selection and hybrid manipulation. If $\mathbf{f}$ indicates the following algebraic expression concerning the identification of the $l$-th T-pattern among those on the left of city centre

$$\kappa_1(\texttt{Trajectories}).west(\pi_{Region}(\sigma_{ID=\text{"}centre\text{"}}(Places))) \quad (\mathbf{f})$$

the overall semantics of $Q_2$ is

$$\mathbf{f}[\![l]\!] \ltimes_{\texttt{contains}} \texttt{Trajectories}$$

Alternatively, the user may be indifferently interested in investigating mobility along T-patterns on the left of the city centre and, hence, ask for all of trajectories therein. $Q_2$ would now be formulated as the following MO-DMQL statement

```
SELECT T.*
FROM   Trajectories T, T_Patterns TS, Places P
WHERE  Places.id="centre",
      (TS.west(Places.Region))⟦·⟧.contains(T.Trajectory)
```

whose semantics follows

$$\mathbf{f}[\![\cdot]\!] \ltimes_{\texttt{contains}} \texttt{Trajectories}$$

**Progressive Mining Tasks.** The possibility of specifying suitable population operators in MO-DMQL allows multiple stages of analysis for the mining results. For instance, to diagnose the causes of mobility congestion, the user may wish to gain an insight into the collective movement of the vehicles in the answer to the last $\mathbf{Q_2}$. In particular, she/he may focus on the ones that move close to one other for longer than 10 time units. The following model-definition statement specifies the discovery of groups of vehicles moving close to each other.

```
CREATE MODEL Moving_Clusters
AS     MINE MC
FROM  (SELECT T.*
       FROM   Trajectories T, T_Patterns TS, Places P
       WHERE P.ID="centre",
             (TS.west(P.Region))⟦·⟧.contains(T.Trajectory))
```

This pattern discovery task involves the execution of the moving cluster algorithm MC, that is one of the schemes in [19]. Notice that the above task specification should be supplemented with a WHERE clause, providing a suitable setting for the algorithm parameters, which is here deliberately omitted, being of no practical relevance for illustration purposes. Procedurally, its meaning is

$$\kappa_2(\mathbf{f}[\![\cdot]\!] \bowtie_{\texttt{contains}} \texttt{Trajectories}) \qquad (\mathbf{s})$$

where $\kappa_1$ is the T-pattern mining algorithm and $\kappa_2$ indicates the adopted technique for unveiling moving clusters.

Despite their complex semantics, population statements provide a simple mean to manipulate data in MO-DMQL. Query $\mathbf{Q_3}$ finds all moving objects in Trajectories, that at time $t$ are inside a moving cluster, whose elements move for at least 10 time units:

```
SELECT Trajectories.*
FROM Trajectories T, Moving_Clusters MC
WHERE (MC.move_for(10))⟦·⟧.inside_at(t,T.Trajectory)
```

Although moving clusters where not formally defined in section 3 for lack of space, in the above statement we adopt them deliberately for illustration purposes. In particular, two suitable pattern operations are exploited. In particular, move_for$(\cdot)$ is a retrieval operation of the composite pattern object MC, which yields a new instance of the *moving clusters* abstract data type. This groups the singletons of $p$ that move together for at least a certain amount of time. Instead, inside_at$(\cdot,\cdot)$ is defined for the individual moving cluster in the collection MC and returns whether a particular moving point of Trajectories resides within MC$[\![i]\!]$ at a given point in time $t$. The semantics of $\mathbf{Q_3}$ in the 2W Model algebra is reported below

$$(\mathbf{s}.\texttt{move\_for(10)})[\![\cdot]\!] \bowtie_{\texttt{inside\_at(t)}} \texttt{Trajectories}$$

where $\mathbf{s}$ succinctly denotes the procedural definition of the foresaid Moving_Clusters model reported above.

## 5. MO-DMQL ENGINE

Hereafter we discuss the design of the DAEDALUS framework, an innovative computational engine, that supports the user in specifying and refining mining objectives as well as combining multiple strategies for analyzing movement data. More specifically, the process of knowledge discovery from moving object data can be seen as an interaction between the data mining engine and the end user, where the latter formulates a query or a statement in MO-DMQL(that describes the patterns of her/his interest) and the former returns the required results.

DAEDALUS supports two extreme types of analytical users: the domain expert and the data mining expert. The former is supported in specifying and refining the analysis goals by means of highly expressive declarative queries. On the contrary, the latter masters the knowledge discovery process and aims at constructing complex vertical analytical solutions and, hence, he/she is supported in specifying and refining the analysis goals by means of procedural abstractions to control the knowledge discovery process.

DAEDALUS implements an object-oriented view of the 2W Model: the raw spatio-temporal data within the D-World and the pattern types within the M-World are kept neatly separated as independent objects and flexibly managed via suitable operators. The engine supports *(i) mining* operators (i.e. mining algorithms), that enable transitions from the D-World to the M-World; *(ii) join* operators, that transition in the opposite direction; *(iii)* world-specific operators, such as *data-from-data* and *model-from-model* operators, that manipulate, respectively, raw data (e.g. for preprocessing purposes) and models (e.g. to meet post-processing requirements).

The query execution engine acts on top of a moving object database, Hermes, an Object-Relational storage layer for persisting both D-World objects and M-World models. D-World objects are simply persisted as instances of some predefined data-type in the rich moving-object data model, natively provided by Hermes. M-World entities depend on the pursued applicative purposes and, hence, require the previous definition of suitable pattern types, to be represented in Hermes.

In DAEDALUS, model definition statements are processed in such a way that the involved mining operators trigger the execution of corresponding mining algorithms over the referred collections of trajectory data. Populate and world-specific operators are instead delegated to Hermes, that is responsible of computing their outcome.

### 5.1 The Hermes Moving Object Database

Hermes [25] is a framework for managing the historical movements of spatial objects, that change their location over time, either discretely or continuously. The framework is meant for providing any extensible Object-Relational DBMSs with spatio-temporal capabilities. Currently, it is developed on a state-of-the-art ORBDMS, i.e. Oracle10g [**?**].

At the heart of Hermes is a Moving Data Cartridge (MDC) [**?**], that combines standard data types in the ODMG object model [**?**] with both the static spatial data types in the *Spatial* Oracle cartridge [**?**] and the temporal literal types within the TAU Temporal Literal Library Data Cartridge [25], to the purpose of defining an extensible spatio-temporal data model. The latter allows to natively represent the D-World entities in terms of Moving_Point, Moving_LineString, Moving_Circle, Moving_Rectangle, Moving_Polygon and Moving_Collection (useful for collections of moving objects), i.e. as moving and morphologically changing object geometries. The individual data types is equipped with a palette of methods, useful for querying moving object properties and relationships.

Depending on the specific applicative goal, the basic data types provided by Hermes can be suitably combined to define composite pattern types, which also enables the representation of the M-World entities. This feature is at the basis of the plug-in architecture of the mining engine in the DAEDALUS framework.

Notably, the integration of the Hermes spatio-temporal data-model within the Oracle ORDBMS results into an intuitive and expressive language for trajectory databases, that essentially extends the Oracle's PL/SQL language with spatio-temporal facilities. Such a language is exploited to support the interactions between DAEDALUS and Hermes, i.e. to store and query both D-World and M-World entities. More specifically, whenever mining and/or populate operators are processed in DAEDALUS, the eventual results (i.e. either new raw data or progressively discovered patterns) are persisted via statements in the extended PL/SQL language. Such statements essentially translate movement data and model objects from the application (i.e. mining algorithms) level into a convenient object-relational representation within the Hermes storage level. In principle, this can be accomplished by adopting one of two alternative design patterns, i.e. either by directly embedding the required scripts within the mining algorithms or via the engineering of suitable stored procedures, providing a centralized access point to the DAEDALUS engine. This latter option was chosen in the design of the DAEDALUS framework, as shown in figure 3, since it confers to the overall framework architecture a high degree of modularity as well as ease of maintenance and still guarantees processing efficiency.

We next provide a taste of the functionality that the Hermes database layer enables in the DAEDALUS framework, by means of an explicative example. Let us suppose that the Trajectories table stores a set of routes. A typical applicative requirement is to filter a subset of such trajectories, fulfilling some desirable properties, upon which to perform a subsequent manipulation task. More precisely, the pursued goal may be to firstly *find those trajectories, that were active between 6am and 2pm on Friday 29/2/2008, to the purpose of retrieving their corresponding sub-trajectories in the specified time period*. Next, the identified sub-trajectories may be exploited to *find those routes that spent more than half an hour inside a certain area, delimited by the two opposite points* $< x_1, y_1 >$ *and* $< x_2, y_2 >$.

In Hermes, the above manipulation process can be performed by issuing two simple statements, $\mathbf{Q_1}$ and $\mathbf{Q_2}$. The former performs a spatio-temporal filtering of the Trajectories data and yields a Morning_Trajectories view, from which the desired results are then obtained through $\mathbf{Q_2}$. Specifically, the spatio-temporal filtering behind $\mathbf{Q_1}$ can be formulated as follows:

```
CREATE VIEW Morning_Trajectories AS
    SELECT T.Trajectory.at_period (
           TAU_TLL.d_period_sec(
           TAU_TLL.D_Timepoint_Sec(2008,2,29,06,00,00),
           TAU_TLL.D_Timepoint_Sec(2008,2,29,14,00,00))
    )
    FROM Trajectories T
```

Notice here the use of the TAU_TLL data cartridge, that provides a clear semantics for the time domain as well as a comprehensive set of temporal types, equipped with related primitives [25]. TAU_TLL is also the name of an object, included in the cartridge, that allows to manage temporal data. In particular, the TAU_TLL.D_Timepoint_Sec primitive defines a generic point in time and is often used in conjunction with the TAU_TLL.d_period_sec facility, that specifies a temporal interval between two time points. This mechanism is used in the example statement $\mathbf{Q_1}$ to filter, starting from the available Trajectories repository, those routes that were active in the required time period.

The desired results can now be obtained by querying the temporal length of the sub-routes in the Morning_Trajectories view, via the foresaid statement $\mathbf{Q_2}$, whose formulation is reported below:

```
SELECT MT.id FROM Morning_Trajectories MT WHERE
        (MT.Trajectory.f_intersection
                (SDO_GEOMETRY(2003, NULL, NULL,
                     SDO_ELEM_INFO_ARRAY(1,1003,3),
                     SDO_ORDINATE_ARRAY(x1, y1, x2, y2))
                ).f_temp_element().duration() > 1800
```

This query exploits SDO_GEOMETRY, the basic spatial data type of the Oracle Spatial Cartridge, that represents the geometry of any spatial object. The latter is specified through an intricate use of several parameters. A discussion of this topic is beyond the scope of the paper. Further details can be found in [?]. In the specific case of the above example, SDO_GEOMETRY is used to represent a bi-dimensional rectangular area, whose opposite ends are represented by the pair of points $< x_1, y_1 >, < x_2, y_2 >$. For each moving object within the Morning_Trajectories view, the Moving_Point.f_intersection primitive identifies the portion of trajectory followed by the object inside the considered area. The temporal interval taken by object to follow the local route is then computed via the Moving_Point.f_temp_element facility. Finally, by means of the TAU_TLL.duration primitive, $\mathbf{Q_2}$ filters those moving points, whose route remained inside the given area for more than 1800 seconds.

The usefulness and applicability of the server-side extensions provided by Hermes into the proposed 2W model framework will be further demonstrated in section 6.

## 5.2 DAEDALUS Anatomy

The software architecture of the DAEDALUS framework, shown in fig. 3, is designed to meet two desirable requirements for any data-mining query-processing system. Firstly, it implements the 2W Model algebraic framework, which enables progressive querying and mining of movement data. Secondly, it processes statements in MO-DMQL, which provides the end user with an intuitive syntax for the definition of the required patterns and a uniform mechanism for handling both movement data and patterns.

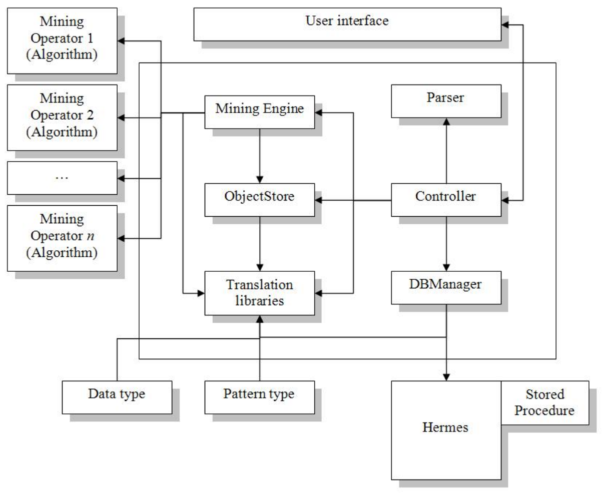

**Figure 3: DAEDALUS anatomy**

The *User interface* component provides the user with a front-end, where to formulate MO-DMQL statements and visualize the corresponding results.

The *Controller* is the core component in the DAEDALUS framework, that is responsible of processing the above statements, via a suitable coordination of the tasks performed by all other components in the framework.

Precisely, the individual statement is validated by the *Parser*, that then converts it into an equivalent OQL syntax, from which a suitable processing plan (i.e. an ordered sequence of steps) is built for answering the statement. Plans are distinguished on the basis of the nature of the statement currently processed. Precisely, statements for data/model querying or manipulation are delegated to the Hermes data layer. Instead, model definition statements involve a very complex management process, that is detailed later.

The *DBManager* provides centralized access to the data layer. Basically, it interacts with the stored procedures in the extended PL/SQL language (mentioned in subsection **??**) in order to store both movement data and patterns. Furthermore, the component exposes facilities for the retrieval of such entities. These facilities transparently perform a complex transformation of any entity from the object-relational representation at the Hermes layer to the object representation at the application level. This is achieved by exploiting the *Translation Libraries*, that essentially convert one representation into other. In practice, the management of a same entity at both the data and application levels requires the addition of suitable wrappers to the *Translation Libraries*, that allow to encapsulate Hermes entities into application-level objects and viceversa.

The *Object Store* is an object repository, that acts as a cache with a twofold purpose: to speed up processing and partially mitigate the object-relational impedance mismatch. The cache transparently makes previously-retrieved data objects available to the mining operator, without re-querying the database layer. For the same reason, it also stores the discovered mining models, before these are persisted within the database layer.

The *Mining Engine* is an environment for the management of a collection of mining algorithms. While processing a given model-definition statement, the *Controller* asks it to execute the one invoked by the `MINE` clause of the statement itself.

The interaction between the above components is discussed below, in the context of the management of model definition statements. As anticipated, these originate very complex processing plans, roughly consisting of the following steps. Initially, the input parameters for the mining algorithms are extracted from the data layer. This step typically involves the retrieval of movement data, whose size can be huge. In such cases, the possible reiterated extraction of the same data, for subsequent processing tasks, can degrade the processing time and, hence, the performance of the overall DAEDALUS framework. To overcome such a shortcoming the extracted data is cached within the *Object Store*, which prevents from subsequently re-querying the database layer. To this point, the *Mining Engine* is activated by the *Controller* to the purpose of executing the mining algorithm invoked in the `MINE` clause of the model definition statement currently processed. The patterns unveiled by the algorithm are not directly stored into the Hermes data layer. Rather, the discovered model objects are cached into the *Object Store*, from where these can be rapidly retrieved by the mining algorithms for possible later processing. Here, the *Controller* converts such model objects into an object-relational representation, by exploiting the *Translation Libraries* and, then, persists them into the Hermes data layer through the *DBManager*.

The results of the processed model definition statement are displayed by the *Controller* on the *User Interface*, by means of an appropriate visualization.

In the overall architecture of the DAEDALUS framework, each component is neatly decoupled from the others and their interactions are dynamically established and governed by the *Controller*, that conforms to a specific coordination protocol, whose role is to release the *Controller* itself from knowing and directly dealing with the inner peculiarities of the individual components. Essentially, the *Controller* knows only the interfaces exposed by the individual components, on the basis of their role in the framework. This guarantees a high degree of modularity and extensibility. Further components with new roles can be integrated within DAEDALUS with minimum effort and maintenance, i.e. by simply specifying their interfaces. As an example, this may be case of either the adoption of multiple user interfaces, to serve different visualization requirements of the end user, or the integration of a plan optimizer, to speed up MO-DMQL statement processing. Also, the existing components can be replaced with distinct ones, as in the case of the adoption of a different moving-object database layer.

Modularity and extensibility are also major aspects of the inner design of the *Translation Libraries* and *Mining Engine* components. Indeed, both support the integration of plug-ins devoted to extend the capabilities of the DAEDALUS framework. Precisely, multiple mining algorithms can be deployed into the *Mining Engine* environment, to the purpose of enlarging the spectrum of the supported mining operators, i.e. of possible patterns discoverable from movement data. In general, the deployment of a mining algorithm requires the definition of a suitable stored-procedure for storing the newly added patterns in some convenient object-relational representation within the Hermes layer as well as the specification of its application-level counterpart. The latter is provided as a *Pattern type* plug-in for the *Translation Libraries*, that also implements an ad-hoc scheme for translating one representation into the other. Besides, further *Data type* plug-ins can be deployed within the same *Translation Libraries* component, for the definition of appropriate object-oriented representations, that are exploited at the application level to encapsulate the raw movement data, resident at the Hermes layer. Both types of plug-ins define suitable wrappers with which to encapsulate both raw data and mining results at the application level. Such wrappers are actually manipulated by the mining algorithms.

As a final remark, in the current implementation of the DAEDALUS framework, a single plug-ins is deployed within the *Mining Engine* environment, that provides an implementation of the T-pattern mining algorithm [5]. Similarly, the *Translation Libraries* comprise a single plug-in, that specifies the object-oriented representation corresponding to the complex `Moving_Point` data type within Hermes.

Modularity and extensibility are also major aspects of the inner design of the *Translation Libraries* and *Mining Engine* components. Indeed, both support the integration of plug-ins devoted to extend the capabilities of the DAEDALUS framework. Precisely, multiple multiple mining algorithms can be deployed into the *Mining Engine* environment, to the purpose of enlarging the spectrum of the supported mining operators, i.e. of possible patterns discoverable from movement data. In general, the deployment of a mining algorithm requires the definition of a suitable stored-procedure for persisting the newly added patterns in some convenient object-relational representation within Hermes as well as the specification of its application-level counterpart. The latter is provided as a *Pattern type* plug-in for the *Translation Libraries*, that also implements an ad-hoc scheme for translating one representation into the other. Besides, further *Data type* plug-ins can be deployed within the same *Translation Libraries* component, for the definition of appropriate object-oriented representations, that are exploited at the application level to encapsulate the raw movement data, resident at the Hermes layer. Both types of plug-ins define suitable wrappers with which to encapsulate both raw data and mining results at the application level. Such wrappers are actually manipulated by the mining algorithms. As a final remark, in the current implementation of the DAEDALUS framework, a single plug-ins is deployed within the *Mining Engine* environment, that provides an implementation of the T-pattern mining algorithm [5]. Similarly, the *Translation Libraries* comprise a single plug-in, that specifies the object-oriented representation corresponding to the complex `Moving_Point` data type within Hermes.

## 6. CONCLUSIONS AND FUTURE WORK

We proposed MO-DMQL, a language for formulating analytical statements with which to progressively mine and query movement data. The procedural semantics of the language is founded on the 2W Model algebraic framework, that allows to accommodate and combine disparate mining tasks into a multi-step knowledge discovery process. Finally, we designed DAEDALUS, an innovative computational engine for processing MO-DMQL statements.

There are some challenging issues, that are worth further research. Foremost, the identification of a compact 2W Model algebra, consisting of a fixed, minimal set of operators. Analogously to the case of the 3W Model framework, this is useful in two respects, i.e. the possibility of expressing the required patterns via suitable combinations of such basic operators, rather than relying on an arbitrary number of task-oriented mining operators, and the development of a solid theoretical background concerning expressiveness and complexity results. Also, the development of strategies for optimizing processing plans would increase the overall performance of the proposed MO-DMQL engine.

## Acknowledgements

This work has been partially supported by the European Community IST-6FP-014915 GeoPKDD project.

## 7. REFERENCES

[1] P.K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. In *Proc. of ACM Symp. on Principles of Database Systems*, pp. 175 – 186, 2000.

[2] L. O. Alvares, et al. A Model for Enriching Trajectories with Semantic Geographical Information. In *Proc. of the ACM Int. Symp. on Advances in Geographic Information Systems*, pp. 162 – 169, 2007.

[3] T. Calders, L. V. S. Lakshmanan, R.T. Ng, and J. Paredaens. Expressive Power of an Algebra for Data Mining. *ACM Transactions on Database Systems*, 31(4):1169–1214, 2006.

[4] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based Most Similar Search. In *Proc. of Int. Conf. on Data Engineering (ICDE'07)*, 2007.

[5] F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli. Trajectory Pattern Mining. In *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages pp. 330 – 339, 2007.

[6] R. H. Güting and M. Schneider. *Moving Objects Databases*. Elsevier, 2005.

[7] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, 2000.

[8] M. Nanni, B. Kuijpers, C. Korner, M. May, and D. Pedreschi. Spatiotemporal Data Mining. In F. GIANNOTTI and D. PEDRESCHI, editors, *Mobility, Data Mining, and Privacy: Geographic Knoweledge Discovery*. Springer-Verlag, 2008.

[9] G. Manco, M. Baglioni, F. Giannotti, B. Kujpers, A. Raffaeta, and C. Renso. Querying and Reasoning for Spatio-Temporal Data Mining. In GIANNOTTI F. and PEDRESCHI D., editors, *Mobility, Data Mining, and Privacy: Geographic Knoweledge Discovery*. Springer-Verlag, 2008.

[10] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-based Languages Journal of the ACM, 42(4):741 – 843, 1995.

[11] O. Wolfson, B. Xuand, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proc. Int Conf. on Scientific and Statistical Database Management*, pp. 111 – 122, 1998.

[12] Jose A. Lema, L. Forlizzi, R. Güting, E. Nardelli, and M. Schneider. Algorithms for Moving Objects Databases. *The Computer Journal*, 46(6):680–712, 2003.

[13] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, M. Schneider, N.A. Lorentzos, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. em ACM Transactions on Database Systems. volume 25, pages 1–42. 2000.

[14] R.H. Güting, V.T. de Almeida, and Z. Ding. Modeling and Querying Moving Objects in Networks. VLDB Journal, 15(2):165 – 190, 2006.

[15] T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. *Commun. ACM*, 39(11):58–64, 1996.

[16] O. Wolfson, A.P. Sistla, S. Chamberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7:257 – 387, 1999.

[17] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and Querying Moving Objects. In W. A. Gray and Per-Åke Larson, editors, *ICDE*, pp. 422–432. IEEE Computer Society, 1997.

[18] T. Johnson, L. Lakshmanan, and R.T Ng. The 3W Model and Algebra for Unified Data Mining. *26th Int. Conference on Very Large Data Bases*, 21 – 32, 2000.

[19] P. Kalnis, N. Mamoulis, and S. Bakiras. On Discovering Moving Clusters in Spatio-Temporal Data. In C. B. Medeiros, M. J. Egenhofer, and E. Bertino, editors, *SSTD*, volume 3633 of *Lecture Notes in Computer Science*, pages 364–381. Springer, 2005.

[20] V. T. Almeida, Ralf Hartmut Güting, and Thomas Behr. Querying Moving Objects in Secondo. In *Proceedings of Mobile Data Management*, page 47, 2006.

[21] S. Gaffney and P. Smyth. Trajectory Clustering with Mixture of Regression Models. In *KDD Conf.*, pages 63–72. ACM, 1999.

[22] J.G. Lee, J. Han, and K.Y. Whang. Trajectory clustering: A Partition-and-Group Framework. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 593 – 604, 2007.

[23] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. In *Addison-Wesley*, 1995.

[24] M. Hadjieleftheriou, G. KolliosĘ, D. Gunopulos, and V. J. Tsotras. On-Line Discovery of Dense Areas in Spatio-temporal Databases. In *Proc. of Int. Int. Symposium on Spatial and Temporal Databases*, pp. 306 – 324, 2003.

[25] N. Pelekis and Y. Theodoridis. Boosting Location-based Services with a Moving Object Database Engine. In *Proc. of Int. ACM SIGMOD/PODS Workshop on Data Engineering*

*for Wireless and Mobile Access*, pp. 3 – 10, 2006.

[26] S. Spaccapietra, et al. A Conceptual View on Trajectories. *Data & Knowledge Engineering*, 65:(1), pp. 126 – 146, 2008.