JTSA: an open source framework for time series abstractions

Lucia Sacchi[a*], Davide Capozzi[b], Riccardo Bellazzi[a], Cristiana Larizza[a]

Department of Electrical Computer and Biomedical Engineering of the University of Pavia, Via Ferrata 5, 27100 Pavia (Italy) (lucia.sacchi@unipv.it,riccardo.bellazzi@unipv.it,cristiana.larizza@unipv.it)

BIOMERIS srl, Via Ferrata 5, 27100 Pavia (Italy) (davide.capozzi@biomeris.com)

* Corresponding author:
Lucia Sacchi
Department of Electrical Computer and Biomedical Engineering of the University of Pavia
Via Ferrata 5, 27100 Pavia (Italy),
email: lucia.sacchi@unipv.it
phone: +390382985981

Abstract.

Background and Objective: The evaluation of the clinical status of a patient is frequently based on the temporal evolution of some parameters, making the detection of temporal patterns a priority in data analysis. Temporal Abstraction (TA) is a methodology widely used in medical reasoning for summarizing and abstracting longitudinal data.

Methods: This paper describes JTSA (Java Time Series Abstractor), a framework including a library of algorithms for time series preprocessing and abstraction and an engine to execute a workflow for temporal data processing. The JTSA framework is grounded on a comprehensive ontology that models temporal data processing both from the data storage and the abstraction computation perspective. The JTSA framework is designed to allow users to build their own analysis workflows by combining different algorithms. Thanks to the modular structure of a workflow, simple to highly complex patterns can be detected. The JTSA framework has been developed in Java 1.7 and is distributed under GPL as a jar file.

Results: JTSA provides: a collection of algorithms to perform temporal abstraction and preprocessing of time series, a framework for defining and executing data analysis workflows based on these algorithms, and a GUI for workflow prototyping and testing.

The whole JTSA project relies on a formal model of the data types and of the algorithms included in the library. This model is the basis for the design and implementation of the software application. Taking into account this formalized structure, the user can easily extend the JTSA framework by adding new algorithms.

Results are shown in the context of the EU project MOSAIC to extract relevant patterns from data coming related to the long term monitoring of diabetic patients.

Conclusions: The proof that JTSA is a versatile tool to be adapted to different needs is given by its possible uses, both as a standalone tool for data summarization and as a module to be embedded into other architectures to select specific phenotypes based on TAs in a large dataset.

Keywords: Temporal abstractions, time series analysis, data analysis workflow, temporal pattern discovery, biomedical data mining software tool

1. Introduction

Temporal Abstraction (TA) is a well known data analysis technique that is frequently applied in clinical domains to analyze complex multivariate clinical histories [1,2], i.e. the series of relevant clinical events occurring to a patient (e.g. hospitalizations, visits, drug intakes, sudden variations of arterial blood pressure and glycemic control). A clinical history is in general made up of a set of temporal sequences related to heterogeneous information. The heterogeneity is not due solely to the variety of clinical variables, but also to the nature of the data (signals, values, reports), to the sampling grid used for data collection and to the temporal granularity of the observations. Since the evaluation of the status of a patient is frequently based on how his/her clinical variables evolve over time, the availability of systems able to automatically detect temporal patterns from data could support physicians in the difficult task of following a large number of patients.

Temporal patterns detection can be particularly useful for a variety of medical analyses, such as data exploration and summarization, temporal reasoning, evaluation of the response to specific treatments, anomaly detection, and prediction of clinical outcomes. Temporal patterns can be extracted using different techniques and several methodologies have been proposed in the literature to achieve this goal. Despite the relevance of TA as a methodology for temporal reasoning, only few efforts have been made to create a framework that is general, easy to use and simple to integrate into other applications.

In this paper we propose JTSA (Java Time Series Abstractor), a Java-based framework providing a library of algorithms for TAs detection, which can be easily extended and integrated into other applications. The novelty of JTSA relies in the possibility of combining different algorithms to define the analysis workflow suitable for detecting the desired temporal patterns. Such patterns can be conveniently extracted from the data thanks to the availability of the JTSA execution engine. A set of simple analysis blocks represents the starting point for building arbitrarily complex workflows. Single blocks and entire workflows are potentially reusable over different data sets and in different applications or medical contexts. The flexibility of JTSA, deriving from the possibility of tuning the values of a set of parameters and of combining blocks into workflows, makes it suitable for a large variety of applications, ranging from chronic diseases management to ICU monitoring. The framework is provided with a GUI that can be used both for workflows prototyping and parameters tuning, and as a tool for offline data exploration.

The paper is organized as follows: Section 2 reviews the related literature on TAs detection tools, Section 3 formally introduces the JTSA analysis process and components, which are then technically detailed in Section 4. Results on the application of JTSA to the diabetes domain are presented in Section 5 and discussed in Section 6.

2. Related Work

Since the early nineties, researchers spent considerable efforts in developing methodologies for temporal reasoning in medicine. Works in this area were mainly oriented to the design of temporal databases and temporal query languages, and to the investigation of algorithms for the extraction of temporal patterns from data.

TA is a methodology particularly suitable for medical applications, as it can be exploited in decision support systems to perform automatic detection of qualitative patterns in longitudinal data [3-11]. It represents an alternative to signal processing techniques in cases when the data have specific features, such as irregular sampling grid or a mixture of qualitative and quantitative variables, which prevent them to be analysed

using traditional methods; as already mentioned, this is rather frequent in clinical monitoring. TA can be used to automatically detect knowledge-based behaviours from data by simulating the way medical experts reason on temporal histories. This allows extending a process that is in general manually carried out on the single patient to the automatic analysis of large patients cohorts.

TA has been first formally introduced as a technique to shift from a time-point representation of quantitative data to a qualitative interval-based description of time series [3]. The first applications of TA were oriented to the detection of patterns that were predefined in specific medical fields and didn't provide a general instrument to be exploited in a wide range of clinical domains. In this context, one of the first decision support tools based on TA is M-HTP [4], a knowledge-based system for monitoring infections in heart-transplanted patients. In this application, an interval-based representation of the patients' clinical history (called Temporal Network) is obtained by extracting TAs from data collected in a clinical database and is then exploited by a rule-based system to provide clinicians with diagnostic and therapeutic suggestions. In this system, complex patterns were used for reasoning purposes, some of them involving multiple time series. Despite the general nature of the implemented algorithms, the fact that they had to be predefined and embedded into the architecture of the system made it hard to re-use them in other applications.

RESUME [12] and its evolution, RASTA [13], are knowledge-based temporal reasoning systems to perform TA. The RESUME system contains a set of temporal reasoning mechanisms aimed at detecting simple TAs, such as State, Rate and Gradient. The types of patterns that can be extracted are modelled in a Knowledge Base. The algorithms to detect such abstractions are characterized by a set of parameters that can be set by the user and are specified in the Parameter Ontology. RASTA is an extension of the RESUME system to take into account larger data sets. This system is embedded in a specific architecture including a temporal database, Chronus II [14], providing a temporal query language. KNAVE-II [15,16] and VISITORS [9,17] are exploration tools based on the RESUME system for TAs extraction. KNAVE-II provides a complex architecture including a knowledge-based visualization and exploration module, a domain ontology server and a temporal-database mediator, and is oriented to visualizing information related to single patients. VISITORS is a tool for intelligent visualization and exploration of time-oriented data of multiple patients. These systems are both focused on the visualization of the extracted temporal patterns and on the exploration of the knowledge base rather than on the data analysis workflow formalization.

TSNet [18] presents a distributed architecture for time series analysis, a collaborative framework specifically designed for sharing algorithms for the abstraction of time series data needed to apply clinical guidelines in a ICU context.

Two papers [19,20] present tools for querying temporal patterns in the data, where the specification of the temporal constraints underlying the query can be performed visually by the user through dedicated interfaces. PatternFinder [19] is a tool for searching and visualizing patterns across multiple patients. With this tool, the user can design the patterns to be extracted using a graphical interface, which is then also used for visualizing the results. In this case, the patterns that can be specified involve pairs of time points constrained on the basis of an adjustable value for the time span between them. More complex patterns are obtained by concatenating simple univariate behaviours using the FOLLOWED-BY relationship. Patterns definition is performed runtime by the user but cannot be saved to be re-used elsewhere. Combi and Oliboni propose a tool [20] for the definition of temporal abstractions based on a technique, the paint strip metaphor, which had already been shown as one of the methods preferred by users to specify patterns [21]. In this work, particular attention is put on the representation of different temporal granularities in the

data [22]. The process for TAs detection is not taken into account in the system, which assumes they are already available from external sources.

A tool with similar functionalities to JTSA is PROTEMPA [8]. This tool is oriented to the extraction of temporal abstractions from large sets of patients to be exploited in retrospective studies. Like JTSA, also PROTEMPA is focused on the definition of a framework for TAs detection, including algorithms specification. Given these analogies, we will provide a more detailed comparison between JTSA and PROTEMPA in the Discussion section. PROTEMPA is exploited into two tools, AIW [23] and EUREKA [24], which are built on top of it. These infrastructures are oriented to phenotype identification based on TAs and other non-temporal data.

3. JTSA Analysis Process

JTSA has been designed as a framework to provide a library of algorithms for time series processing and abstraction, and a standalone application for the definition and execution of a complete time series analysis workflow. The whole JTSA project relies on a formal model of the data types and of the algorithms included in the library. This model has been exploited as a basis for the design and implementation of the software application. Taking into account this formalized structure, the user can easily extend the JTSA framework by adding new algorithms. In this section we introduce the methodological TA framework underlying JTSA and we will detail the building blocks that allow carrying out the analysis process.

3.1 Methodological Ontology

JTSA methodological framework is based on the ontology shown in Figure 1.

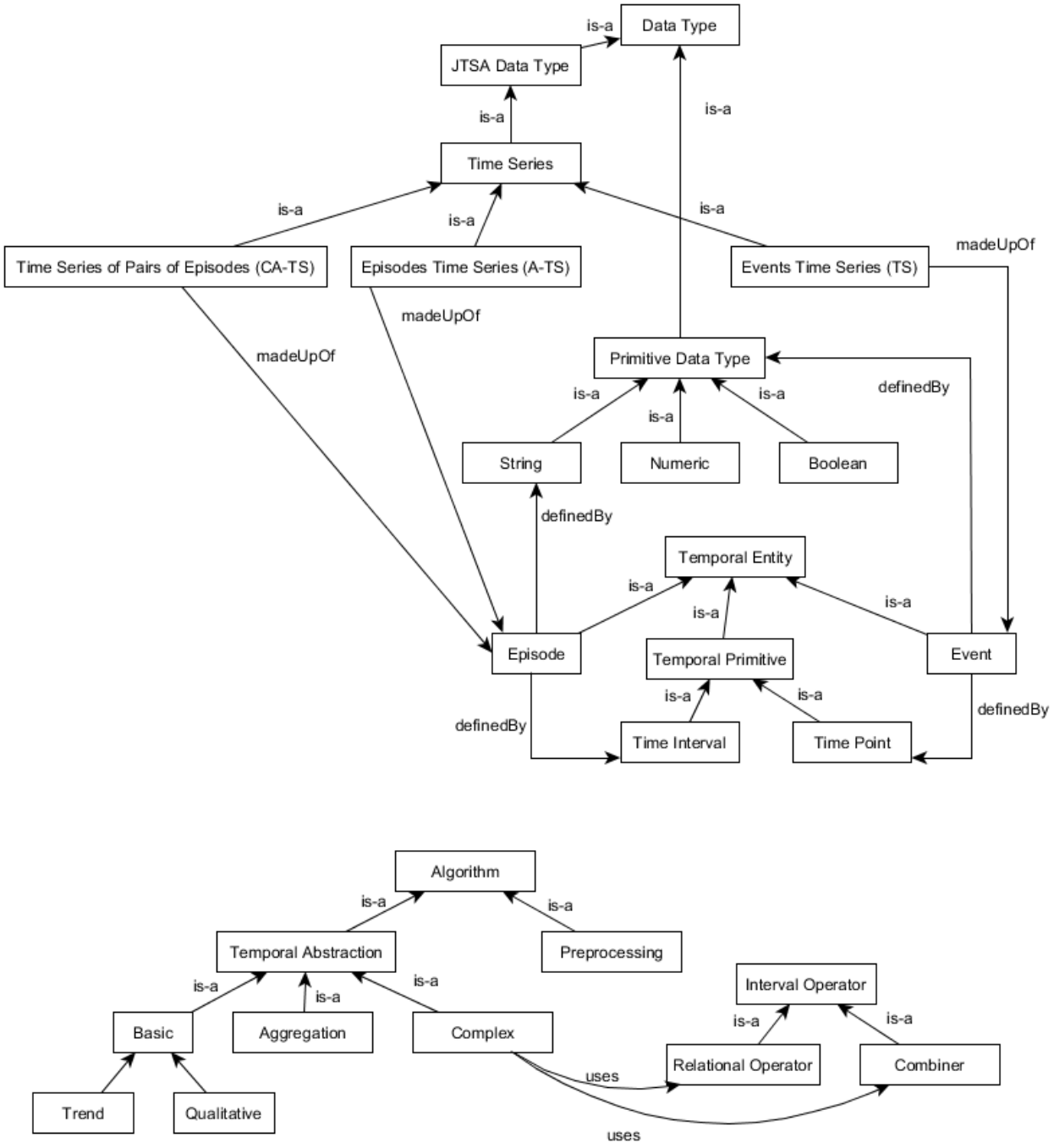

Figure 1. JTSA methodological ontology

This ontology, including the taxonomies describing temporal entities, data types, algorithms and temporal operators, is inspired by the temporal abstraction framework first presented in [4] and later formalized in [25]. With respect to this framework, two new concepts have been introduced, Aggregation and Combiner, both needed for reasoning with time intervals.

In the JTSA framework, we consider two temporal primitives: the *time point*, defined by a timestamp *t*, and the *time interval* (*I*), defined by an ordered pair of time points $[t_{start}, t_{end}]$, where $t_{start} <= t_{end}$. The maximum temporal resolution used to represent a time point or a time interval is known as *granularity* [26]. On the

basis of the selected granularity, the temporal information can be either associated to a time point, and denoted as *event*, or to a time interval, thus denoted as *episode*. Each e*vent* is characterized by a pair (*t, v*) where *t* is the time point of occurrence and *v* is the value associated to the measure of a parameter (it can be of any type, e.g. numeric, boolean, string). Raw data collected over time usually come in the form of events. An *episode* is characterized by a pair (*I, L*), where *I* is the interval of occurrence of the episode and *L* is the label associated to the temporal pattern that holds during *I*. In our framework, we will use episodes to represent the results of TA.

Relying on these temporal entities, it is possible to define the main data types that can be managed using our framework: *events time series* (TS) and *episodes (abstractions) time series* (A-TS). As shown in Figure1, we have also introduced a concept to represent *time series of pairs of episodes* (CA-TS), which result from applying temporal operators to episodes.

The *Algorithm* taxonomy includes both algorithms for time series preprocessing and algorithms for TA detection. *Preprocessing* is an important step in time series analysis. Thus, as the purpose of the framework is to provide a flexible instrument relevant to a wide range of applications, we have included also these algorithms in the tool. The main feature of preprocessing algorithms (that differentiates them from TAs) is that they work on a TS and produce a TS as the output. On the other hand, TA algorithms always produce A-TSs starting from either TSs or A-TSs. The preprocessing algorithms currently implemented in the framework allow filtering, smoothing, normalization and interpolation of numeric time series. As a matter of fact, there exist also other algorithms that might be included in this category, for example to deal with symbolic time series or to implement more sophisticated filtering techniques. As we will discuss in more detail in Section 4, new algorithms can easily be added to the framework, thus allowing the user to have all the instruments needed for the analyses of interest.

The *Temporal Abstraction* taxonomy has several levels. The first level includes three types of algorithms: Basic, Aggregation and Complex. These algorithms differ on the input data type and on the number of inputs, as detailed in Table I. *Basic* TA takes a TS as input and outputs an A-TS. Basic TA algorithms can be of different types, depending on the pattern they detect in the data. *Qualitative* TA performs a mapping between ranges of quantitative values and qualitative levels, which are then outputted as A-TSs where episodes are zero-length intervals. *Trend* TA detects decreasing, increasing or stationary patterns in numerical TS. The output of Trend TA is an A-TS where the episodes correspond to the intervals of validity of the pattern in the data.

Table 1. Algorithms input and output

Algorithm	Input	Output
Preprocessing	TS	TS
Basic	TS	A-TS
Aggregation	A-TS	A-TS
Complex	Pair of A-TS	A-TS

Aggregation TA takes as input one A-TS and outputs another A-TS where consecutive episodes are merged according to a set of rules that are specific to each algorithm. Examples using Aggregation TA will be shown in the Results Section.

Complex TA can be used to detect complex patterns in univariate or multivariate data. These patterns are defined through the application of one of the *Relational Operators* and one of the *Combiners* provided in the library. Relational Operators, including Allen's relationships [27], are binary operators. For this reason, Complex TA works on pairs of A-TSs. Relational Operators generate CA-TS, which are the series of the pairs of intervals verifying the temporal relation specified by the TA and make up the input for the Combiner. To properly derive these intervals, the relational operators are characterized by a set of parameters that allow constraining the mutual temporal position of the output CA-TS elements (details on these parameters are available in [25]). Combiners define how each pair of CA-TS intervals has to be processed to obtain a single output interval for the resulting A-TS. Examples of Combiners are the union or the intersection of two intervals.

3.2 Workflow Components

One of the novel features of the JTSA framework is that it allows users to build their own analysis workflow by combining different algorithms. In our processing model, a workflow is made up of a set of components (blocks), each embedding one or more algorithms. Figures 2 and 3 show the JTSA blocks, their internal components and the input/output data types.

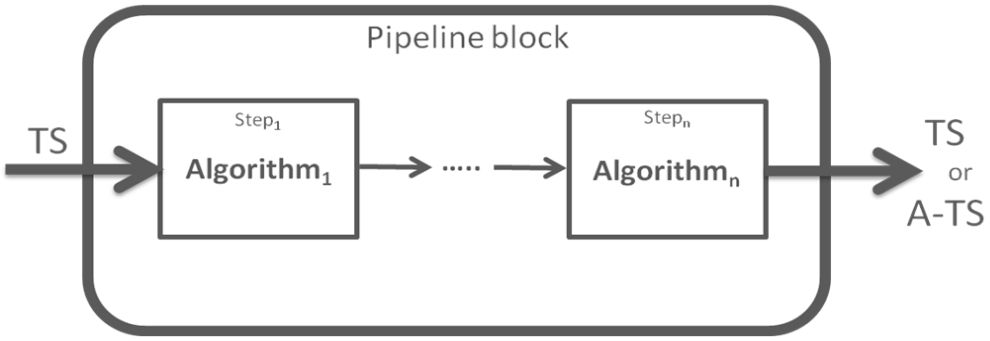

Figure 2. Pipeline block

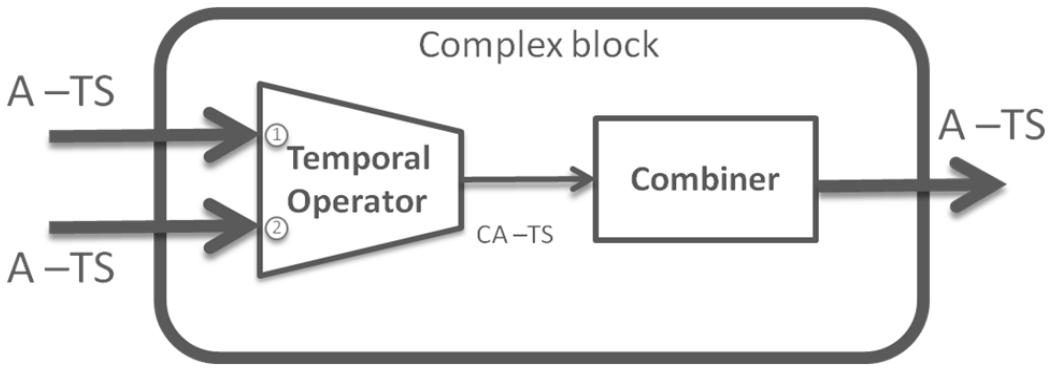

Figure 3. Complex block

Depending on the number and type of inputs (Table 1), we distinguish Pipeline and Complex blocks. Pipelines (Figure 2) contain an ordered set of Steps, which are the components used to implement a single Preprocessing, Basic or Aggregation algorithm. Since the steps are arranged in sequence, to define a pipeline the user has to provide the steps ordering, taking into account that the output data type of each step must be compliant with the input data type of the following one. Complex blocks (Figure 3) are used to

implement Complex TA algorithms. Each complex block allows extracting one type of pattern relying on a single relational operator and one combiner.

A block can take as input either the output of another block or data coming from external data sources, acquired through dedicated components, which we have called Readers. The output of each block can be provided to the user through a component called Renderer, which can be optionally added to the workflow. Currently available Renderers are: the standard output, a file or a graph.

By combining Pipelines and Complex blocks the user can create workflows to extract arbitrarily complex patterns. The workflow is saved in a XML file according to a specific schema, as we will detail in Section 4. The workflow structure can be defined using the GUI of the standalone application or manually, by writing the corresponding XML file.

3.3 Engine

The JTSA Engine is the component responsible for running the analysis workflow. Its main task is the execution of a sequence of blocks, after having established their order using a specific algorithm. Each block b_i is characterized by a unique identifier (ID_i) that must be used by any other block b_j to specify that the input to b_j comes from b_i. The engine uses these IDs to reconstruct the execution sequence of the blocks in the workflow. The blocks whose input is taken from a Reader are the first to be executed. An error message is generated if the engine cannot reach a complete ordering of the blocks list (e.g. if no blocks that take their input from an external source are available or if a loop is identified). Moreover, the engine is in charge of checking that the output of one block is compatible with the input to the next. As already mentioned, the ordering of the steps that make up a Pipeline block is explicitly specified by the user during pipeline definition. Once blocks are correctly ordered, the workflow execution starts and continues by considering the blocks sequentially. Running the algorithm(s) included in a block is the core phase of its execution.

The extraction of meaningful patterns depends on the customization of a set of specific parameters (e.g. maximum slope for trend abstractions, maximum gap between adjacent intervals for Aggregation, etc.). These parameters are provided by the user in a separate properties file whose name is specified during workflow definition in the XML file or in the GUI (see the example in Section 4.1). Since many of these parameters serve to control some temporal constraints, for the majority of the algorithms it is necessary to specify also the granularity, i.e. the time unit adopted for the algorithm parameterization. In the current version of JTSA the granularity parameter can assume one of the following values: SECONDS, MINUTES, HOURS and DAYS. Besides the file containing the parameters, each algorithm is also associated to a meta-parameters file defining the name, type and multiplicity of each parameter. The allowed parameter types are: integer, double, boolean and string. The possible multiplicities are: scalar and vector. The engine uses the meta-parameters file to perform a formal validation of the parameters specified in the properties file, before the algorithm is executed.

4. The JTSA Framework

The JTSA framework has been developed in Java 1.7 and is distributed under GPL as a jar file. Third-party libraries have been used for graphical renderers implementation (JFreeChart http://www.jfree.org/jfreechart/) and for trend detection (Weka library [28]). The java classes/interfaces structure reflects the JTSA methodological ontology and the workflow structure presented in Section 3. Every algorithm is embedded into a class suitably included in the JTSA classes hierarchy. The currently

available algorithms derive from methodologies previously prototyped in other programming languages (C and Matlab) and exploited in other applications [29,30,31]. The present version of the JTSA library contains 5 Preprocessing algorithms, 9 Basic TA algorithms, 2 Aggregation, 14 Relational Operators and 11 Combiners.

4.1 JTSA Use Cases

To illustrate the JTSA framework and highlight the most important features of the tool, we rely on the use cases taken into account during JTSA design and implementation. For illustrating these use cases, we will consider an example related to the detection of a simple pattern: finding intervals where body temperature is increasing, assuming that the parameter is regularly measured every 6 hours. This pattern, which we will label as “INCREASING”, can be detected using a Basic TA, since the input is the numerical TS of body temperature measurements and the output is a time series of episodes where the measurements show a monotonic increase. To detect this pattern, the user will have to build a workflow that simply considers one Pipeline block including a single step implementing a Basic Trend TA (Figure 4). The output is an A-TS containing the sequence of increasing episodes, as shown in Figure 5. In the following, we will describe in details how it is operatively possible to define (and execute) this workflow using the JTSA framework.

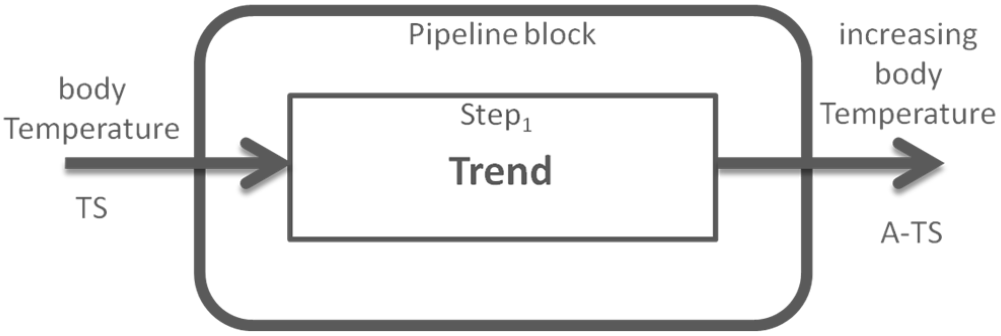

Figure 4. Workflow for the definition of the "INCREASING" body temperature pipeline

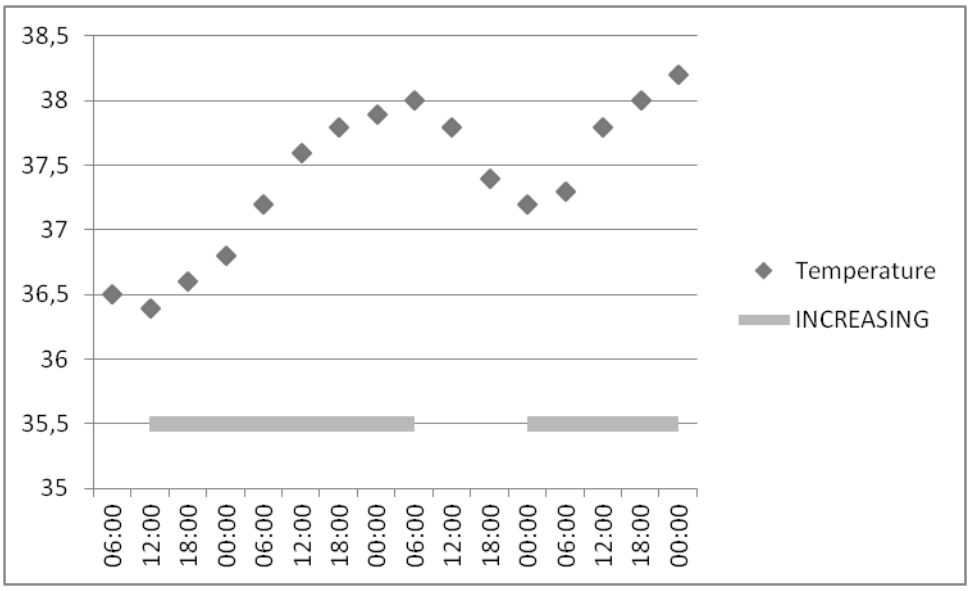

Figure 5. Output of the "INCREASING" body temperature workflow

In particular, we will consider the following three scenarios:

1. definition and execution of the workflow using the JTSA GUI;
2. creation of an XML file containing the workflow definition. The workflow can then executed by a program exploiting the JTSA API;
3. extension of the JTSA library by adding a new algorithm for pattern detection. The algorithm is then executed in the workflow as described in the previous use cases.

The JTSA GUI (use case 1) provides the easiest way for defining and executing an analysis workflow, as the user can specify the complete sequence of blocks by filling in a set of forms. Figure 6 shows how these forms have been filled in for the “INCREASING" body temperature pattern. From the initial GUI window (menu File), the user can select the type of blocks to be included in the workflow (Pipeline or Complex Block). The user can also load a predefined workflow from an XML file, previously generated either using the GUI or manually.

In the top panel, the user has to specify the name of the input data file and the type of reader (in this example we chose a Reader for a CSV file). When a Pipeline block is selected, the user can create the desired number of steps by using the button placed in the lower part of the window ("add new step to the pipeline"). Each step has its own tab in the central panel of the window. For each step, the user has to specify: the name of the step, the order of execution within the pipeline, and the algorithm that will be used to process the time series. These algorithms are classified into Preprocessing, Basic or Aggregation. Each of these categories includes the set of specific algorithms available in the library. In our example, a Basic TA type (BASIC) is selected and a sliding window algorithm is chosen to perform trend detection (BASIC_TRENDSLIDINGWINDOW). Once the algorithm has been selected, the user has to indicate the name of the file where its parameters are listed. These parameters must comply with the meta-parameters defined for the specific algorithm. For the sliding window trend detection algorithm in the example, the meta-parameters specification is:

```
name = gap, minLen, granularity, label, minSlope, maxSlope
type= Integer, Integer, String, String, Double, Double
multiplicity= Scalar, Scalar, Scalar, Scalar, Scalar, Scalar
```

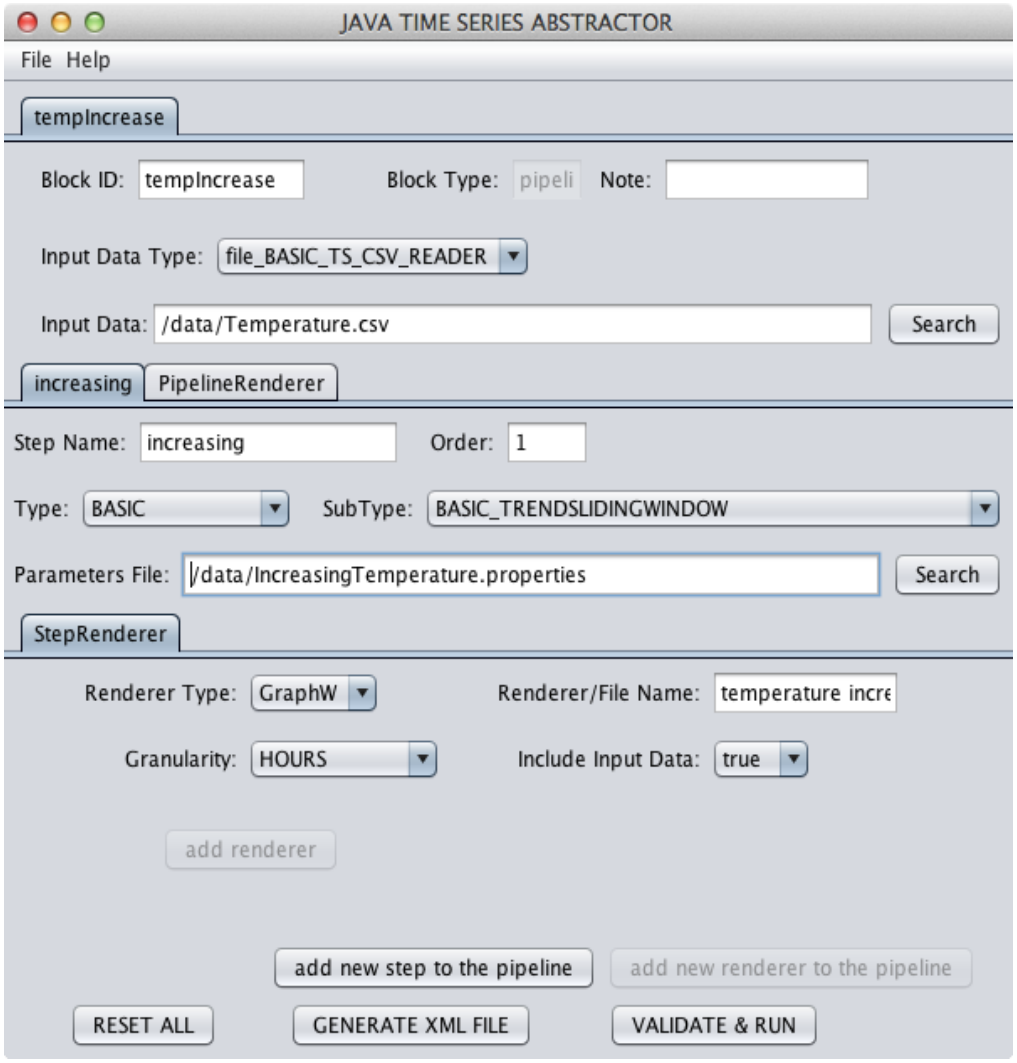

Figure 6. JTSA GUI for workflow definition

The parameters file provided for running the algorithm in this example has the following structure:

```
! gap: maximum distance between two consecutive measurements in a pattern
gap=8
! minLen: minimum length of a pattern
minLen=1
! granularity: granularity of the provided TS
granularity=HOURS
! label: pattern label
label=Increasing
! minSlope: minimum slope allowed in the pattern
minSlope=0.1
! maxSlope: maximum slope allowed in the pattern
maxSlope=+1
```

In this example, lines starting with '!' denote comments. These parameters values allow extracting an increasing pattern, either slow (`minSlope`>=0.1°/hour) or fast (`maxSlope` <=1°/hour), lasting at least 1 hour (`minLen`=1 and `granularity`=HOURS) and aggregating measurements whose distance is not greater than 8 hours (`gap`=8). Parameters values are validated before execution to verify they fulfil the algorithm specific constraints (e.g. `minLen` must be a positive value).

To visualize the input and/or the output of the step, a graphical renderer is selected (bottom panel). In case a pipeline made up of more than one step is defined, the user can visualize partial outputs using a Step Renderer and the final output using a Pipeline Renderer. From the GUI, the user can also save the workflow definition in an XML file for future re-use, using the button “GENERATE XML FILE”.

An alternative way of defining a workflow is the manual creation of the XML file, according to a schema that is provided in the JTSA library (Use case 2). The XML file defining the simple workflow shown in the example above is the following:

```
  <run>
      <block id="tempIncrease" type="pipeline" note=""
          dataType1="file_BASIC_TS_CSV_READER"
          dataIn1="/data/Temperature.csv">
          <step order="1" title="increasing"
              parameters="/data/IncreasingTemperature.properties"
              type="BASIC" subtype="BASIC_TRENDSLIDINGWINDOW">
             <renderer type="GraphW" label="temperature increase"
                  granularity="HOURS" renderInput="true"/>
          </step>
      </block>
</run>
```

In this case, the user has to provide the name of the algorithm type and subtype, the input data file and the name of the file where the parameter values are specified.

When the XML file has been created, the workflow can either be loaded and executed from the GUI, or it can be directly run from a user defined program.

The third use case allows exploiting the JTSA infrastructure to add custom algorithms to the library. In the case of trend detection, the user might for example be interested in defining a new strategy to extract trends or in specifying an additional algorithm to preprocess the input time series before pattern detection. Another possibility is adding new Readers for specific input data formats. In this use case, the user will have to implement the java class corresponding to the desired algorithm, inserting it in the appropriate package selected by considering the methodological ontology. Moreover, the meta-parameters file will have to be created, to guide future users in parameters specification and setting. The engine will use this file for validation (see Section 3.3). Intuitively, this use case is more suitable for expert users since it requires programming skills to develop the algorithm to be integrated into the library. When a new algorithm is added to JTSA, the GUI will automatically make it available among the analysis options.

4.2 Workflow prototyping with JTSA

Workflow prototyping is a very useful feature of JTSA, which can be conveniently exploited to define a set of patterns potentially re-usable in different applications and on different data sets. In this paragraph we illustrate how to use JTSA to design these types of workflows and save them for future use. This issue can be easily addressed using the JTSA GUI, as shown in use case 1. Once the workflow structure has been defined, the user can run it directly on his/her data, properly adjusting the algorithms parameters, if needed.

Figure 7. Patterns available with the JTSA library

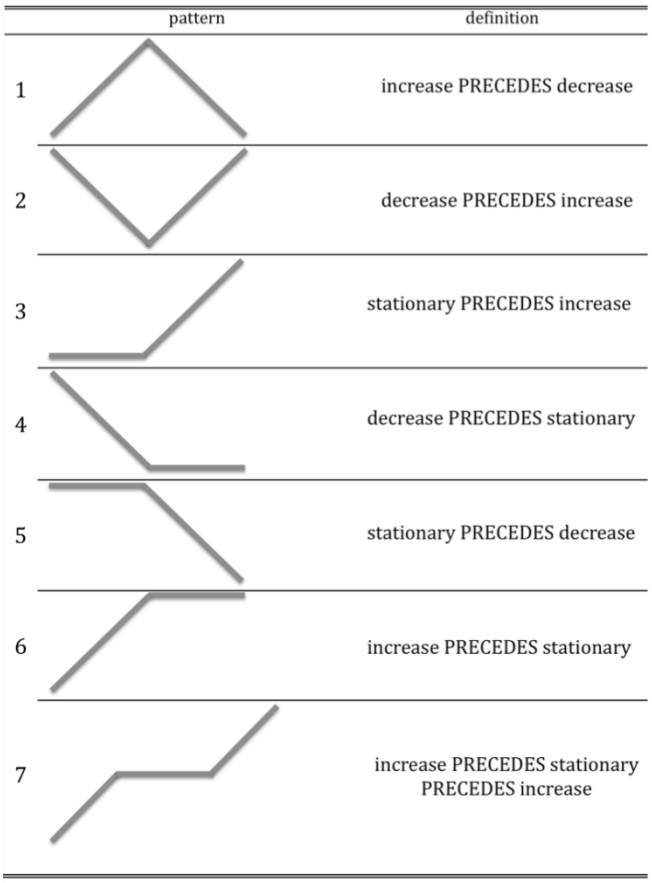

Figure 7. Patterns available with the JTSA library

Figure 7 reports the patterns that are currently provided together with the JTSA library as a collection of XML files. Pattern 1 identifies an up-and-down behaviour in a time series, which can be qualitatively interpreted as a "peak". Pattern 2 is a “reverse peak” (down-and-up behaviour). Pattern 3 represents an increase of the values of a variable after a stable period, whereas pattern 4 represents a variable decreasing to then reach a stable value. Patterns 5 and 6 represent, respectively, a decrease after stable values and an increase to reach a plateau. Finally, pattern 7 shows a complex behaviour where an increasing trend is interrupted by a stable period. In the figure we show both a qualitative picture of the patterns and their definition through the TA formalism. These patterns can be formalized by simply combining three Pipeline blocks, used to detect increase, decrease and stationary patterns, and a Complex block, exploiting the PRECEDES temporal operator[1] [5] and the Union Combiner[2]. As an example, Figure 8 shows the workflow to extract a peak in a variable (Pattern 1). It is important to point out that in certain cases, the same pattern can be detected in different ways. For example a peak can be extracted also using the MEETS operator. Moreover, these workflows can take as input more than one time series of different type (numeric or categorical) allowing the extraction of multivariate patterns.

During the prototyping phase it is useful to inspect the plot of the input TS and of the intervals of the resulting output patterns, using some testing data. Figure 9 shows the plots obtained using the JTSA GUI to execute the workflow in Figure 8 on the body temperature time series of a test patient. Here the pattern

[1] Given two intervals $I_1 = [t_{start1}, t_{end1}]$ and $I_2 = [t_{start2}, t_{end2}]$, I_1 PRECEDES I_2 iff $t_{start1} <= t_{start2}$ ^ $t_{end1} <= t_{end2}$

[2] I_1 Union I_2 results in the interval $[t_{start1}, t_{end2}]$

has been extracted using a single variable (temperature), but the workflow can be applied to two different variables to extract a multivariate complex pattern (see Section 5.4 for an example).

The JTSA library and GUI are available at the following link: http://biomeris.com/documents/software.

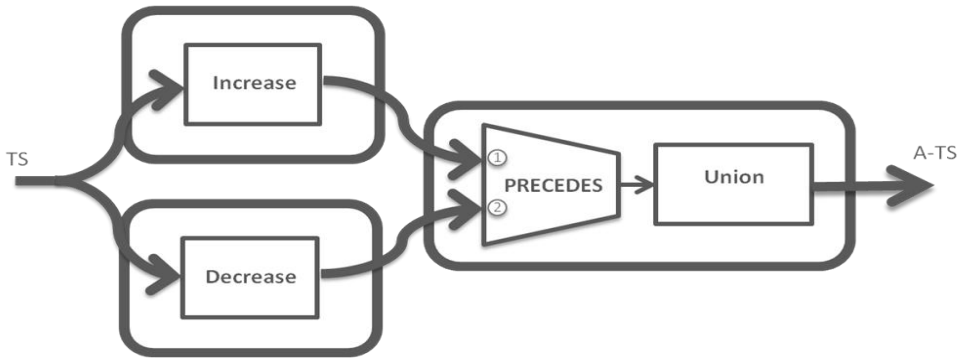

Figure 8. Example of workflow to extract a peak pattern on a single variable

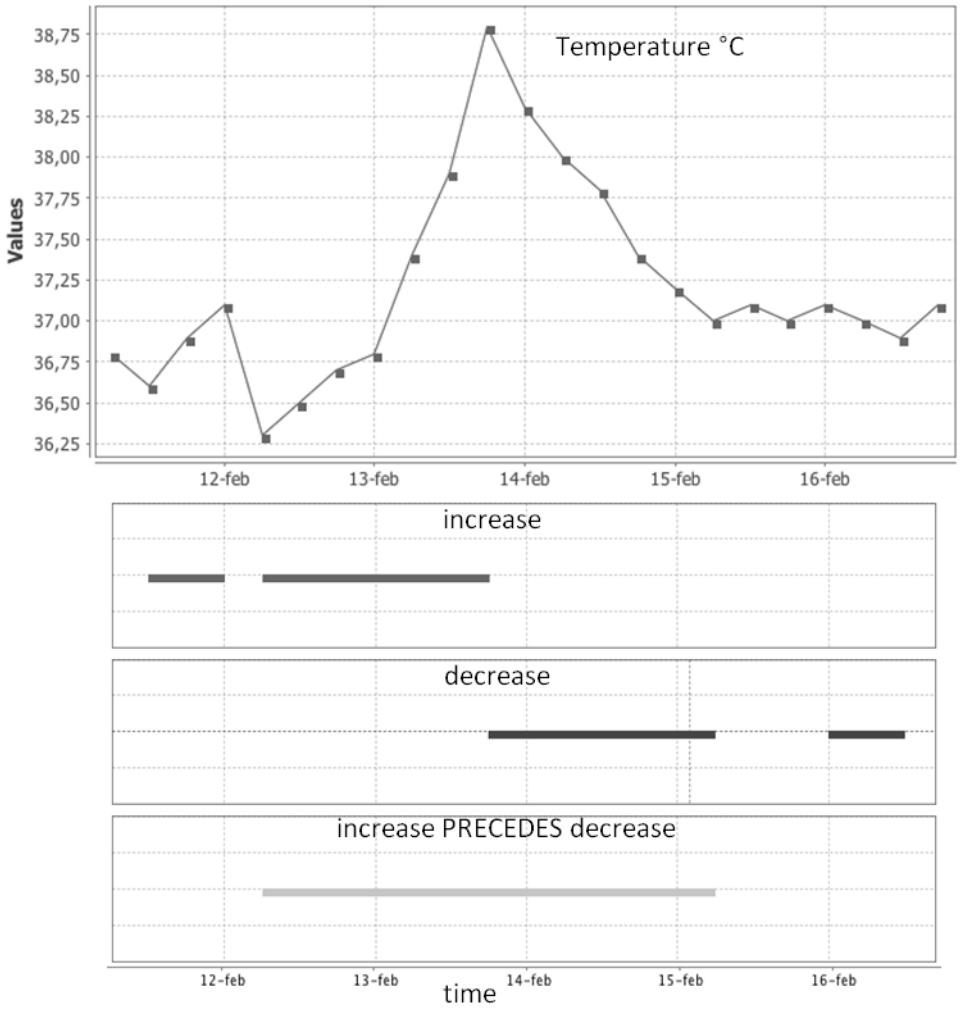

Figure 9. Extraction of a peak in body temperature. The first graph shows the input time series, the second shows the intervals where the temperature is increasing, the third shows the intervals where the temperature is decreasing. The last graph shows the resulting peak interval extracted by the TA detection algorithm

5. Results

We have applied the JTSA framework to the extraction of temporal patterns from data related to diabetic patients collected in the EU project MOSAIC (http://www.mosaicproject.eu/). The MOSAIC project is devoted to the development of models and methodologies to enhance the currently available tools for the diagnosis and management of diabetes. The project is focused on the identification of a set of risk profiles, characterized by clinical and environmental factors and by specific patterns of care, able to stratify the population with the objective of delivering a more targeted and personalized care. Data coming from the hospitals participating to the consortium are stored in the i2b2 data warehouse (DW) (https://www.i2b2.org/software [32]) and record the clinical histories of the patients for a period of 15 years. To synthesize the temporal behaviour of both clinical parameters and life style variables, we have included in the DW also a set of TAs. These TAs have been defined thanks to a close collaboration with the medical experts, who provided the domain knowledge on the most important temporal patterns they usually look for in the data. According to their suggestions, the variables that have been processed for TA detection were: diet, HbA1c and weight. Diet data are collected at each follow-up visit as a categorical value that can be “good” or “bad”. Clinical parameters such as weight and HbA1c are collected as numerical variables with a frequency that depends on the general conditions of the patient. It is important to point out that one of the strengths of TA applied to clinical domains is that it can be easily exploited to deal with irregularly sampled data. Regarding diet, physicians are interested in evaluating intervals where a patient shows good eating habits. As regards weight, physicians are interested in identifying both the time intervals where patients loose weight and the time that takes to the weight to reach a specific target. Weight loss thanks to good eating habits is important for all diabetic patients, as it has been shown in the literature to lower cardiovascular risk [33]. As regards HbA1c, clinicians were interested in investigating the amount of time that patients take to reach a specific target level.

In the following of this section we will illustrate how the medical knowledge was translated to create suitable analysis workflows in JTSA. In addition, we will present the results of the execution of these workflows on a sample dataset. This will highlight the most important features of the framework.

5.1 Abstractions on diet

The extraction of intervals of good eating habits is a Basic abstraction task and highlights the capability of JTSA of running on data of different types. As already mentioned, diet is a categorical variable assuming “good” or ”bad” values. The workflow that allows detecting the desired pattern, is made up by a single pipeline block containing two steps. First, the qualitative TS needs to be adapted to the suitable JTSA data structure as A-TS and, second, an Aggregation TA algorithm merges consecutive “good” episodes to create clinically meaningful patterns.

5.2 Abstractions on weight

Finding intervals of decreasing weight requires the same workflow as the one shown in Figure 4. More interesting is to describe the procedure that leads to the definition of the workflow to extract the time needed to reach a specific weight target (time-to-target). The “TIME-TO-TARGET” pattern can be seen as

the interval lasting from the first out-of-target value to the moment a patient reaches a specific weight target. To extract such pattern using a JTSA workflow, we need to perform the following stages (Figure 11):

1. Find “OUT-OF-TARGET” episodes
2. Find “IN-TARGET” episodes.
3. Find “OUT-OF-TARGET” episodes followed by “IN-TARGET” episodes
4. Extract the output “TIME-TO-TARGET” episodes

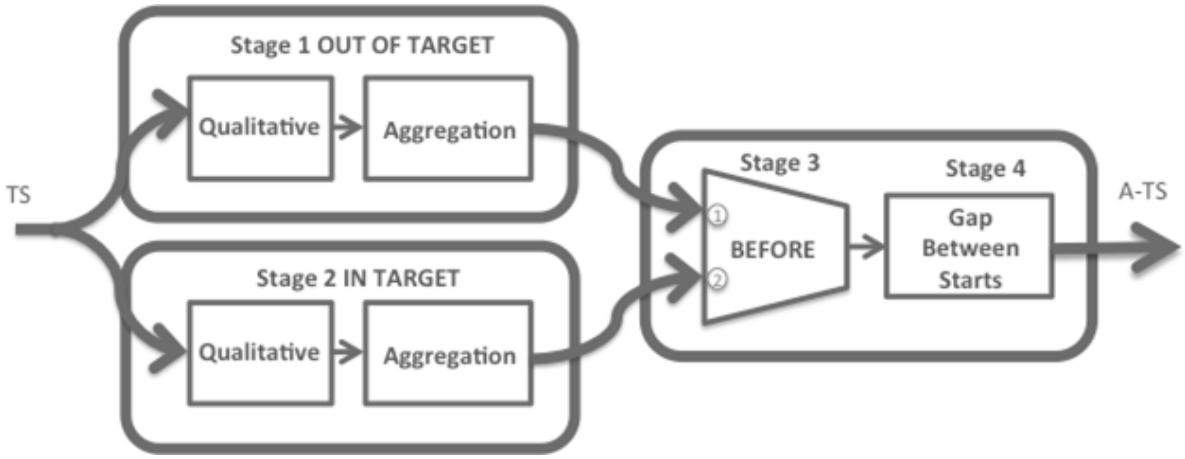

Figure 10. Workflow for computing the “TIME-TO-TARGET” abstraction on weight

Stages 1 and 2 can be performed using two Pipeline blocks including Basic and Aggregation abstraction algorithms. The target weight loss is patient-specific and has been set to the 10% of the baseline patient’s weight, in accordance to the literature [33].

The first Pipeline in Figure 10 detects “OUT-OF-TARGET” episodes, whereas the second one detects “IN-TARGET” episodes. Stages 3 and 4 are performed by a Complex block. In particular, stage 3 is obtained by applying the BEFORE temporal operator to the input episodes. It provides a set of pairs of episodes that the Combiner (stage 4) will process to generate the final episodes.

Figure 11 shows how the Complex TA is computed, given a set of “IN TARGET” and “OUT OF TARGET” input episodes. Figure 11.a shows the “OUT OF TARGET” (A-TS_1) and “IN TARGET” (A-TS_2) episodes time series that are the input to the Complex block. Figure 11.b shows the result of the BEFORE temporal operator applied to the inputs. The BEFORE operator looks for pairs of intervals $I_i = [t_{starti}, t_{endi}]$ and $I_j = [t_{startj}, t_{endj}]$ that verify the relationship $t_{endi} < t_{startj}$. The output is the series of pairs of episodes (CA-TS) where the relationship holds. Finally, the Combiner called “GapBetweenStarts“ extracts the “TIME-TO-TARGET” episodes defined on the interval $[t_{starti}, t_{startj}]$. The output episodes time series in Figure 11.c results: A-TS_{out} = {([t_1, t_3], “TIME-TO-TARGET”), ([t_5, t_7], “TIME-TO-TARGET”)}. For relational operators, a set of parameters is available to constrain the maximum distance between the intervals in a pair. In the figure, for example, suitable values for these parameters are set to prevent interval [t_1, t_2] to be related to interval [t_7, t_8]. A more detailed description of these parameters can be found in [25].

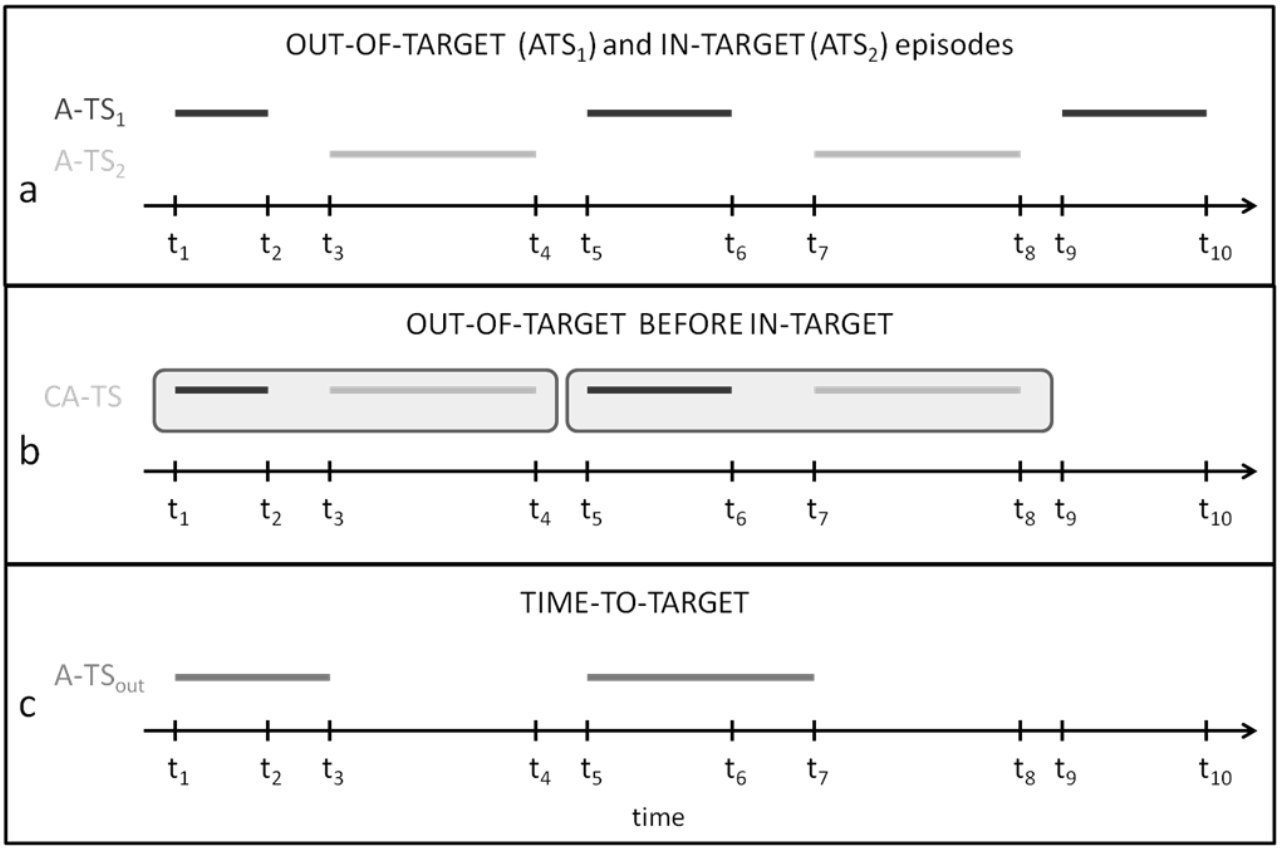

Figure 11. Workflow for computing the ”TIME-TO-TARGET” abstractions

5.3 Abstractions on HbA1c

The workflow components to extract the time taken to a patient to reach a specific HbA1c level are the same as the ones described for weight in the previous paragraph. In the case of HbA1c, the target level is defined as a value that does not depend on the baseline HbA1c value recorded for each patient. Thus, the only difference stands in the algorithm selected to perform the first step of the two Pipeline blocks.

5.4 Abstractions on weight and diet

To show how JTSA can be applied to heterogeneous data, we consider a complex pattern on weight and diet data. This pattern involves numerical and categorical data and is intended to extract time intervals where the weight is decreasing and the diet is good. This pattern could identify periods of good compliance to diet recommendations that result in a decrease of body weight. Finding intervals of decreasing weight and good eating habits requires the workflows described in sections 5.1 and 5.2. The “GOOD-COMPLIANCE” pattern has been defined as a Complex TA where the PRECEDES operator is applied to intervals of good eating habits and intervals of decreasing body weight. The “Union” Combiner extracts the “GOOD-COMPLIANCE” episodes. Figure 12 depicts the workflow to detect the “GOOD-COMPLIANCE” pattern.

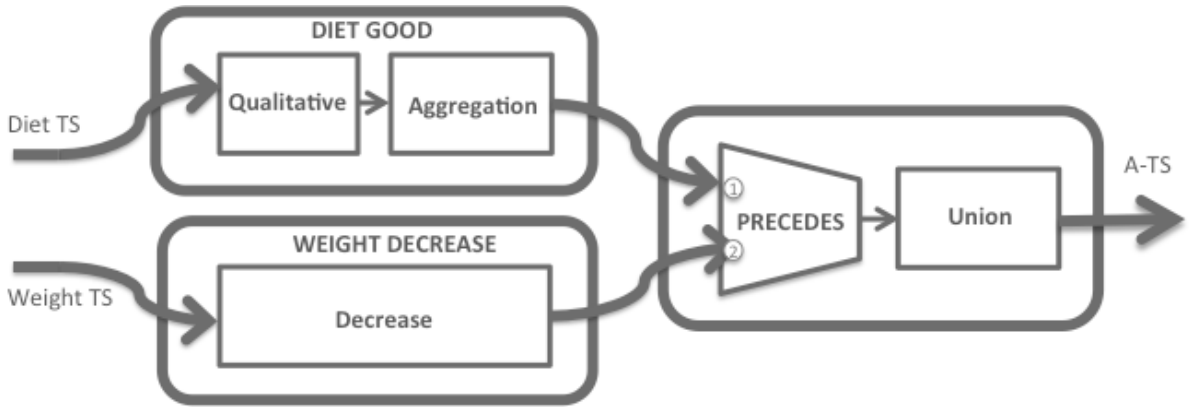

Figure 12. Workflow for computing the "GOOD-COMPLIANCE" patterns

5.5 Extending JTSA with MOSAIC-specific algorithms

An important feature of the JTSA framework is the possibility of adding new algorithms according to the types defined in the ontology and invoking them in a workflow that can be run through the GUI or by implementing a customized program.

In this context, two new algorithms have been purposely implemented to deal with the MOSAIC scenarios. The first algorithm is able to detect a decrease in a numerical TS, given the decrease rate and the temporal horizon of the decrease (e.g. decrease of 10% in 6 months), instead of its absolute value (e.g. decrease of 1 kg/month). This new algorithm exploits the available sliding window trend detection mechanism by simply setting the slope parameter to a custom value computed on the basis of the specific input TS. Similarly, we have developed a second algorithm as an extension of the Basic Qualitative TA algorithm, where the threshold is patient-specific and is calculated based on a specific value of the input TS (e.g. target defined as the 90% of the baseline value of the specific variable).

5.6 Applying JTSA to the MOSAIC data

The MOSAIC project is aimed at analysing multivariate data related to a set of selected patients. Since the JTSA core components are designed to work at the single patient and workflow level, we needed to define an additional module (called JTSA Wrapper) to automatically run a set of workflows on several patients. We have applied the JTSA wrapper to the data stored in the MOSAIC DW to extract the desired patterns.

Table 2 shows a summary of the workflow components needed to extract the patterns defined for the MOSAIC project, as described in sections 5.1, 5.2, 5.3 and 5.4 of this paragraph.

Table 2. Workflows details for the MOSAIC TAs

Workflow name	Number of steps	Pipelines	Complex
WeightTimeToTarget	4	2	1
WeightDecreasing	1	1	0
DietBad	2	1	0
DietGood	2	1	0
HbA1cTimeToTarget - 7	6	2	1
HbA1cTimeToTarget - 7.5	6	2	1
HbA1cTimeToTarget - 8	6	2	1
GoodCompliance	3	2	1

The available dataset includes 444 subjects, followed for a period of fifteen years by the Fondazione Salvatore Maugeri Hospital located in Pavia, Italy. Table 3 reports, for each variable, the total number of collected time points and the average number of time points per patient. The JTSA Wrapper module, during the execution of the workflows, produces a log file reporting some information about the analysed data and the extracted patterns. In the final part, a summary report is provided. It includes details on: input and output file names, total number of patients, overall analysis period, number and list of the executed workflows, overall number of extracted patterns, date of the analysis and total elapsed execution time.

Table 3. Summary of the data collected for each variable

Variable	Total number of time points	Average number of time points per patient
Diet	5554	6.25
Weight	5558	12.5
Hba1c	4834	10.9
Total	15946	35.9

The following summary report is generated when extracting the patterns specified in Table 2:

```
=======================================
      Summary Report
=======================================
input: dati/exportMosaic.txt
output: dati/outputCPaperMosaic.txt
---------------------
Data set
---------------------
patients: 444
period: [09/03/1999 00:00:00 - 28/04/2014 00:00:00]
---------------------
Analysis results
---------------------
workflows: 8
=======================================
 # patterns  # patients      pattern
```

```
=======================================
 50     |   46     |    WeightTimeToTarget
=======================================
 790    |   354    |    WeightDecreasing
=======================================
 458    |   248    |    HbA1cTimeToTarget-7
=======================================
 434    |   226    |    HbA1cTimeToTarget-7.5
=======================================
 243    |   172    |    HbA1cTimeToTarget-8
=======================================
 475    |   317    |    DietGood
=======================================
 473    |   265    |    DietBad
=======================================
 39     |   36     |    GoodCompliance
=======================================
2962 patterns
---------------------
Execution info
---------------------
start date: 23/01/2015 15:52:44
elapsed time: 28.321146 seconds
=======================================
```

6. Discussion

The analysis of clinical data often requires considering information that is heterogeneous both for its representation (quantitative, qualitative, categorical, etc.) and for its temporal connotation (static or time varying). Medical reasoning is frequently based on the evaluation, matching, and comparison of qualitative knowledge-based temporal patterns; these are usually investigated to summarize the clinical status of the patient for diagnostic, therapeutic or prognostic purposes. Temporal abstraction allows performing this task automatically, offering a way to synthesize heterogeneous information into a homogeneous representation using intervals and labels. Although a number of tools have been proposed in the literature to implement TA mechanisms, only few of them can be seen as general instruments that can be easily integrated into other systems. JTSA has been conceived as a comprehensive tool that allows dealing with temporal information by abstracting it into patterns. For the definition of the JTSA framework, we have designed a methodological ontology, which is able to describe clinical data processing both from a data storage and an abstraction computation perspective. JTSA provides:

- a collection of algorithms to perform temporal abstraction and preprocessing of time series,
- a framework for defining and executing data analysis workflows based on these algorithms,
- a GUI for workflow prototyping and testing.

Taking into account the data types and algorithms defined in the JTSA ontology, it is possible to extend the library with new algorithms that can be included in a workflow to be executed by the engine. Thanks to the fact that an arbitrary number of blocks can be used within a workflow, a wide range of patterns can be detected, from simple to highly complex. A specific feature of JTSA, which distinguishes it from other available tools, is that potentially any kind of pattern can be defined and retrieved. This variety of patterns is even wider if we consider tuning the parameters of a single algorithm. Finally, the same workflow can be re-used and applied to different data sets, provided that the input data sources are specified and their format is compliant with the selected reader.

The system that has more similarities with JTSA is PROTEMPA, a modular architecture for temporal abstraction including a framework for TAs definition and extraction [8]. It relies on an Algorithm Source dedicated to primitives definition and on a Knowledge Source storing the algorithms parameters. The main difference between our tool and PROTEMPA lays in the definition of the analysis process model. We have formalized the idea of the workflow, as a standardized structure based on blocks. This modularity allows creating complex workflows potentially including parallel pipelines or just complex blocks, which is not possible using the PROTEMPA framework. The most commonly used workflows can be included in a library, which could be re-used on different data sets and conditions. Moreover, we have the possibility of introducing pre-processing steps in the analysis workflow, either before applying TA algorithms or by their own to perform time series transformation. In addition, the analysis process formalization includes the engine, which is the module in charge of validating and executing the workflow. Once the workflow has been specified (e.g. in XML format), it is straightforward to create the code to run it using the engine. In this paper, this feature has been exploited to integrate temporal abstraction in the MOSAIC project. DWs as the one used in this project support only basic temporal query, and an instrument to compute temporal patterns can provide some advanced data mining capabilities.

The current version of JTSA has some limitations, both related to the possible usage scenarios (in particular through the GUI) and to the computational optimization. As a matter of fact, the framework is still oriented to the analysis of a single patient and to the execution of a single workflow. In case it is necessary to perform an analysis on a patients set using multiple workflows, a custom application has to be implemented, as we have shown in the case of the MOSAIC Wrapper. Moreover, the GUI is thought as a tool for workflow prototyping and parameters tuning rather than as a query tool for extracting patterns in patients datasets. Even if the tool is directed to physicians, the workflow design phase is still quite complicated, especially when the patterns to be extracted are complex and involve multiple variables. In these cases, the same behaviour can even be detected using different workflows. For this reason, the choice of the sequence of blocks in the workflow and their parameterization could require the support of a knowledge engineer. For this reason, JTSA has recently been integrated into the i2b2 framework, as a plugin to deliver a user friendly tool mainly oriented to physicians. As a matter of fact, the i2b2 framework currently supports only temporal query but lacks a processing tool able to dynamically extract arbitrarily complex temporal patterns. The availability of the plugin has allowed a preliminary usability analysis to evaluate how users (physicians and computers scientists) perceive the JTSA framework. As soon as the plugin will be deployed and made available to the MOSAIC clinical partners, it will be possible to get a useful feedback on possible additional algorithms that might be added to the library. Furthermore, an enhancement of the standalone application based on graphical metaphors could improve the usability of the system.

Regarding the performance of a workflow execution, the current version of the algorithm in charge of establishing the order of blocks to be run does not recognize the blocks that might be run in parallel. As a matter of fact, the engine is designed to run independent blocks separately. In the perspective of using the tool to query large datasets, an extension towards a parallel execution would be a high-priority task. Parallelization might be exploited to run in parallel independent blocks within the same workflow, to run in parallel different workflows and to run in parallel the same workflow on different patients.

7. Conclusion

Although TA is a methodology frequently used in medical reasoning for longitudinal data exploration and abstraction, only few attempts have been made to provide general tools for TA to be effectively exploited

in different clinical contexts. JTSA is a framework that has been developed with a twofold aim: first, to supply an extensible library of algorithms for TAs, and, second, to provide an engine able to run a workflow combining different algorithms. The definition and execution of a workflow allows the extraction of arbitrarily complex patterns from longitudinal data. The proof that JTSA is a versatile instrument to be adapted to different needs is given by its possible uses, both as a standalone tool for data summarization and as a module to be embedded into a complex architecture to select specific phenotypes based on TAs in a large dataset.

Acknowledgements

This work was supported by the EU funded project MOSAIC, part of the 7-th Framework Program (FP/-600914). The authors would like to thank Luca Fiorina for developing part of the framework and Mauro Bucalo for supporting data warehouse integration. We would also like to thank Prof. Luca Chiovato, Pasquale De Cata and Paola Leporati of the Fondazione Maugeri Hospital for their precious contribution in TAs definition for the MOSAIC project.

References

1. K. Orphanou, A. Stassopoulou, E. Keravnou, Temporal abstraction and temporal Bayesian networks in clinical domains: a survey, Artif. Intell. Med. 60(3) (2014) 133-49.
2. M. Stacey, C. McGregor, Temporal abstraction in intelligent clinical data analysis: a survey, Artif. Intell. Med. 39 (2007) 1–24.
3. Y. Shahar, A framework for knowledge-based temporal abstraction, Artificial Intelligence. 90(1-2) (1997) 79–133.
4. C. Larizza, A. Moglia, M. Stefanelli, M-HTP: A system for monitoring heart transplant patients, Artif. Intell. Med. 4(2) (1992) 111-126.
5. R. Bellazzi, C. Larizza, P. Magni, R. Bellazzi, Temporal data mining for the quality assessment of hemodialysis services, Artif. Intell. Med. 34(1) (2005) 25-39.
6. R. Bellazzi, C. Larizza, A. Riva, Temporal abstractions for interpreting diabetic patients monitoring data, Intelligent Data Analysis. 2(1-4) (1998) 97–122.
7. M. Verduijn, L. Sacchi, N. Peek, R. Bellazzi, E. de Jonge, B.A. de Mol, Temporal abstraction for feature extraction: a comparative case study in prediction from intensive care monitoring data, Artif. Intell. Med. 41 (2007) 1–12.
8. A.R. Post, J.H. Harrison, PROTEMPA: A Method for Specifying and Identifying Temporal Sequences in Retrospective Data for Patient Selection. J. Am. Med. Inform. Assoc. 14 (2007) 674–683.
9. D. Klimov, Y. Shahar, M. Taieb-Maimon, Intelligent visualization and exploration of time-oriented data of multiple patients. Artif. Intell. Med. 49(1) (2010) 11-31.
10. L. Sacchi, C. Larizza, P. Magni, R. Bellazzi, Precedence Temporal Networks to represent temporal relationships in gene expression data. J. Biomed. Inform. 40(6) (2007) 761-74.
11. R. Bellazzi, C. Larizza, P. Magni, S. Montani, M. Stefanelli, Intelligent Analysis Of Clinical Time Series: An Application In The Diabetes Mellitus Domain. Artif. Intell. Med. 20 (2000) 37-57.
12. Y. Shahar, M.A. Musen, RESUME: a temporal-abstraction system for patient monitoring. Comput Biomed. Res. 26(3) (1993) 255-73.

13. M.J. O'Connor, W.E. Grosso, S.W. Tu, M.A. Musen, RASTA: a distributed temporal abstraction system to facilitate knowledge driven monitoring of clinical databases. Medinfo 2001:508 –12.
14. M.J. O'Connor, S.W. Tu, M.A. Musen, The Chronus II Temporal Database Mediator. Proc. AMIA Symp. (2002) 567–571.
15. Y. Shahar, D. Goren-Bar, D. Boaz, G. Tahan, Distributed, intelligent, interactive visualization and exploration of time-oriented clinical data and their abstractions. Artif. Intell. Med. 38(2) (2006) 115-35.
16. S.B. Martins, Y. Shahar, M. Galperin, H. Kaizer, D. Goren-Bar, D. McNaughton, L.V. Basso, M.K. Goldstein, Evaluation of KNAVE-II: a tool for intelligent query and exploration of patient data. Stud. Health Technol. Inform. 107(Pt 1) (2004) 648-52.
17. D. Klimov, Y. Shahar, M. Taieb-Maimon, Intelligent selection and retrieval of multiple time-oriented records. Journal of Intelligent Information Systems 35(2) (2010) 261–300.
18. J. Hunter, TSNet - a distributed architecture for time series analysis. Stud. Health Technol. Inform. 139 (2008) 223-32.
19. J. Fails, A. Karlson, L. Shahamat, B. Shneiderman, A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events across Multiple Histories. IEEE Symposium On Visual Analytics Science And Technology (2006) 167- 174.
20. C. Combi, B. Oliboni, Visually defining and querying consistent multi-granular clinical temporal abstractions. Artif. Intell. Med. 54(2) (2012) 75-101.
21. L. Chittaro, C. Combi, Visualizing queries on databases of temporal histories: new metaphors and their evaluation. Data & Knowledge Engineering 44(2) (2003) 239–64.
22. C. Combi, M. Franceschet, A. Peron, Representing and reasoning about temporal granularities. Journal of Logic and Computation. 14(1) (2004) 51–77.
23. A.R. Post, T Kurc, S. Cholleti, J. Gao, X. Lin, W. Bornstein, D. Cantrell, D. Levine, S. Hohmann, J.H. Saltz, The Analytic Information Warehouse (AIW): A platform for analytics using electronic health record data. J. Biomed. Inform. 46(2) (2013) 410-24.
24. A.R. Post, T. Kurc, R. Willard, H. Rathod, M. Mansour, A.K. Pai, W.M. Torian, S. Agravat, S. Sturm, J.H. Saltz, Temporal abstraction-based clinical phenotyping with Eureka! AMIA Annu. Symp. Proc. 16 (2013) 1160-9
25. L. Sacchi, C. Larizza, C. Combi, R. Bellazzi, Data Mining with Temporal Abstractions: Learning Rules from Time Series. Data Mining and Knowledge Discovery, 15 (2007) 217-247.
26. C. Bettini, X.S. Wang, S. Jajodia, A general framework for time granularity and its application to temporal reasoning. Annals of Mathematics and Artificial Intelligence. 22(1-2) (1998) 29–58.
27. J.F. Allen, Towards a general theory of action and time. Artif. Intell. 23 (1984) 123–154.
28. I.H. Witten, E. Frank, Data Mining: Practical Machine Learning. Tools and Techniques, second ed. Morgan Kaufmann Series in Data Management Systems, 2011.
29. C. Larizza, R. Bellazzi, G. Lanzola, An Http Based Server For Temporal Abstractions. Idamap '99 Working Notes (1999) 52-62.
30. L. Sacchi, R. Bellazzi, C. Larizza, P. Magni, T. Curk, U. Petrovic, B. Zupan, Ta-Clustering: Cluster Analysis Of Gene Expression Profiles Through Temporal Abstractions. Int. J. Med. Inform. 74 (2005) 505-517.
31. P. Magni, F. Ferrazzi, L. Sacchi, R. Bellazzi, TimeClust: a clustering tool for gene expression time series. Bioinformatics. 24(3) (2008) 430-2.
32. S.N. Murphy, G. Weber, M. Mendis, V. Gainer, H.C. Chueh, S. Churchill, I. Kohane, Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). J. Am. Med. Inform. Assoc. 17(2) (2010) 124–130.
33. J.B. Meigs, D.M. Nathan, R. B. D'Agostino, P.W.F. Wilson, Fasting and Postchallenge Glycemia and Cardiovascular Disease Risk: The Framingham Offspring Study. Diabetes Care 25(10) (2002) 1845-1850.