DETECTING DEPENDENCIES BETWEEN STATES OF MULTIPLE DATA STREAMS

Abstract: The subject of dependency detection has been addressed many times during the last 20 years, and methods that extract frequent patterns or temporal information from data have been developed. The information extracted has been used in fields like medicine, network surveillance, stock analysis and many others, thus proving its importance in real life. But as technology evolved, the amount of generated information grew to the point where storing it for later analysis was no longer an option. In this paper we propose a method of temporal dependency detection between states of heterogeneous data streams, that processes the information as it arrives and discards it afterwards, and we test it on real surveillance data.

Keywords: *Data mining, stream mining, temporal dependency detection.*

I. INTRODUCTION

In recent years, the evolution in hardware technology led to a significant drop of prices for monitoring equipment, thus enabling even small scale companies to acquire and use such devices. The data produced by these devices is usually stored and then analyzed in order to automatically detect anomalies or to extract associations that are hard to observe by a human operator. But as the amount of collected data increases every day, the storage becomes expensive and the amount of data gets harder and harder to analyze in a timely fashion.

This is why researchers are currently working on developing data mining techniques that are capable of extracting information in a single pass of the data. These modern techniques work on infinite sequences of events, ordered in time, called data streams.

The detection of temporal dependencies between states of data streams is a type of frequent pattern mining.

Throughout the history different researchers proposed various time representation techniques that can be used to express temporal information.

Allen proposed in [1] the Allen's relations for temporal intervals, a subset of which are used in this paper to express types of transitions. The same relations are presented in [2], complemented by logic describing how such information should be represented and extracted from data sets. In [3], Freksa describes a set of relations for semi-intervals which can be used in cases where there's not enough knowledge to express the information using Allen's relations. Recently, a summary of time representation methods and mining techniques has been published by Fabien Moerchen in [4]. Here we can find additional methods used to represent time, like Roddick's relations or the unification-based temporal grammar proposed by Ultsch. For our purpose a small subset of Allen's relations are sufficient.

Most of the existing frequent pattern mining algorithms need multiple passes in order to analyze a data set and to produce relevant information. This is the case with:

- The Apriori algorithm, [5], which represents the root of a class of algorithms having the same name. This algorithm has also been used recently in order to extract frequent activities within the CASAS project, [6].
- MSDD, [7], and its sibling MEDD, [8], which try to detect if a set of characteristics can predict another. They start from a basic structure which is refined through multiple iterations.
- The Apriori Stochastic Dependency Detection, ASDD, which has Apriori and MSDD at its base, but does not take into account the duration when searching for patterns, [9].
- PrefixSpan, [10], SPADE, [11], Armada, [12], and the algorithm described in [13] are other algorithms that require multiple passes in order to mine data. The representation of data used in this paper is somewhat similar to the one used when describing the Armada algorithm.

However, algorithms that work on data streams do exist, but they are not applicable to our problem. The papers consulted include:

- [14], where Hoefding Trees are defined and an algorithm that uses them to extract information from data streams is proposed. This is not applicable as duration is not taken into account during analysis.
- [15] and [16], that describe algorithms for analysis of network data in one pass, but that do not take into account the temporal aspect of the information.

Other consulted papers include [17], where much of the work presented focuses on the adaptation of known data mining techniques to data stream processing, [18] and [19] where the primary constraints related to data stream processing have been defined, like the requirement to cope with high speed events, concept drifting and other problems that are inherent, like adaptability to failures within data sources.

Mining high speed data streams also implies the use of continuous processing techniques. The development of such techniques started with [20], [21] and [22], where the

Manuscript received February 29, 2010; revised April 17, 2010

notion of dataspaces was defined together with a couple of principles that should drive future development of such systems in the right direction. The development in a "pay-as-you-go" fashion is proposed, in order to allow the addition of semantics corresponding to a data source over time. The main feature of a dataspace support system is the possibility to interpret newly added sources in a dumb way and to allow querying in a non-standard way until semantics are added for them [19].

Continuous query systems were developed before the notion of dataspaces was coined. These systems have an architecture similar to what has been described in the aforementioned papers and hence there use while designing an architecture for the continuous processing of data streams.

Telegraph [23], CACQ [24], TelegraphCQ [25], Cayuga [26] and SASE [27] are all systems that allow processing of queries over incoming data streams. Queries are specified using query languages specifically defined for these systems. They feature caching modules and operators that are optimized for parallel processing. The joining of information from multiple sources is performed using join operators that use in memory caches. Cayuga allows the publishing of queries as new streams, a feature which permits the definition of complex queries. The experience gained while developing these systems can be used to perform enhancements on current stream mining systems.

In this paper we approach the problem of dependency detection between data streams and the problem of input stream modeling.

The time dependency between states is relevant in modern monitoring systems because it allows the construction of a graph of causality and they ease the job of anomaly detection.

A dispute might occur relating to dependency detection stating that manual rules could be defined in a system where the relations between sources are known. While this is perfectly true, the manual definition of rules is error prone and can be cumbersome to implement on system employing a large number of sources.

The remainder of this paper is organized as follows: section II presents the main concepts we are going to work with, section III presents the problem that is going to be addressed, sections IV and V present the proposed solution and the results obtained after testing it, and the section VI concludes the paper.

II. DEFINITIONS

A. Data streams

A *data stream* DS is a sequence of time stamped data events produced by a data source ds. The data event is a tuple containing a timestamp and the attributes of the state to which the data source transitioned. We assume that a data source cannot be in more than one state at a time (unique timestamp) and consider the timestamps as a monotone sequence of positive integers (discrete-time).

$$DS = \{(t_i, a_i^1, \dots, a_i^k) \mid a_i^j \in A^j, t_i \in \mathbb{N}, t_i < t_{i+1}\} \quad (1).$$

Within relation (1), t_i represents the timestamp, a_i^j is the value of the data source attribute a^j at the moment t_i, A^j is the set containing all possible values of the j^{th} attribute $\forall j = \overline{1,k}$ and $k \in \mathbb{N}^+$ is the total number of attributes of the data source.

Let $DS_1, DS_2, \dots, DS_n$ be the data streams we work with, where n represents the number of streams.

B. Data stream states

Definition. A state s_a is a unique combination of values of the source attributes and it is represented as a tuple that takes values from the Cartesian product $A^1 \times \dots \times A^k$:

$$s_a = \{(a_{s_a}^1, \dots, a_i^k) \mid a_i^j \in A^j, t_i \in \mathbb{N}, t_i < t_{i+1}\} \quad (2).$$

Within relation (2), $a_{s_a}^1$ is the value of the a^1 attribute for the state s_a, $a_{s_a}^1 \in A^1$.

We assume that the list of attributes used to describe a source does not change over time.

A data source is in a state s_a each time the values of the source attributes are equal to the values describing the state. Based on the definition of a data stream we can conclude that a data stream is in the state corresponding to the latest data event until another newer event is generated.

Example. Considering an electric switch we can perform the following associations:

- A state *ON* can be associated to a power switch between the events ***switch on*** and ***switch off***.
- A state *OFF* can be associated to the same switch between the events ***switch off*** and ***switch on***.

The figure 1 bellow illustrates the relation between events and states.

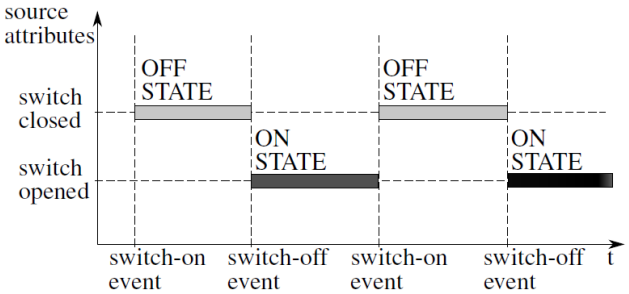

Figure 1. Events and associated states for a switch.

When referring to states that belong to multiple data sources the name of the data stream will be specified before the attributes in the tuple, like this: $s_a = (DS_1, a_{s_a}^1, \dots, a_{s_a}^K)$, or as an exponent of the state $s_a^{DS_1}$. The short notation s_a should only be used when the data source (stream) can be directly inferred from the context.

The set of all states that belong to a data source DS_1 is:

$$S(DS_1) = \bigcup_{i=\overline{1,m_1}} s_i \quad (3),$$

where m_1 is the number of possible states for DS_1.

C. Occurrences of states

Definition. An occurrence oc_i of a state $s_a \in S(DS_j), \forall j = \overline{1,n}$ is a maximal time interval within which the source (stream) is in the specified state. The occurrence can be represented as a triple, as within formula (4). In the same formula $state(ds,t)$ is a function that returns the state of the source ds at the moment t.

$$oc_i^{s_a} = \left(s_a, t_{start}^{oc_1}, t_{end}^{oc_1}\right), t_{start}^{oc_1} < t_{end}^{oc_1}, \nexists\, t_{start}^{oc_1} < t < t_{end}^{oc_1}\ s.t.\ state(ds,t) \neq s_a \quad (4)$$

The notation oc_i is denoted as the short notation and should only be used when the state and data stream can be inferred from the context.

Example. We illustrate the notion of occurrences, within figure 2, using a data stream DS and the states s_a, s_b and s_c. Note the fact that the state s_a occurs multiple times within the data stream.

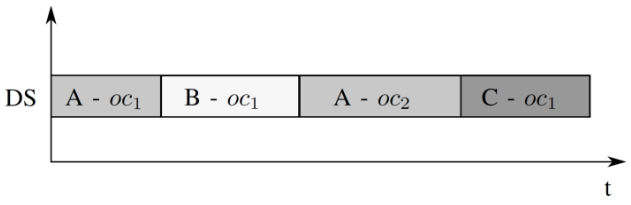

Figure 2. Occurrences of states on a data stream.

As a state s_a can occur multiple times within a data stream DS_j, it is necessary to define the set containing all occurrences of a state within that data stream:

$$OC\left(s_a^{DS_j}\right) = \bigcup_{i=\overline{1,p_j}} oc_i \quad (5),$$

where p_j is the number of occurrences of the state s_a within the data stream DS_j.

D. Transitions

Let $s_a \in S(DS_1)$ and $s_b \in S(DS_2)$ be two states and $oc_i(s_a, t_{start}^{oc_a}, t_{end}^{oc_a})$ and $oc_j(s_b, t_{start}^{oc_b}, t_{end}^{oc_b})$ be occurrences of the states.

Definition. A transition tr between occurrences $oc_i^{s_a}$ and $oc_j^{s_b}$ is defined as a tuple $tr(oc_i^{s_a}, oc_j^{s_a})$, formed by the states s_a, s_b and a combination of the parameters $\Delta t_{start\_tr}^{a\to b}$, $\Delta t_{end\_tr}^{a\to b}$, $gap_{tr}^{a\to b}$,$rgap_{tr}^{a\to b}$, defined within the group of relations 7.

$$tr\left(oc_i^{s_a}, oc_j^{s_a}\right) = \left(s_a, s_b, \Delta t_{start\_tr}^{a\to b}, \Delta t_{end\_tr}^{a\to b}, gap_{tr}^{a\to b}, rgap_{tr}^{a\to b}\right) \quad (6),$$

where:

$$\Delta t_{start\_tr}^{a\to b} = t_{start}^{oc_j^{s_b}} - t_{start}^{oc_i^{s_a}} \quad (7.1),$$

$$\Delta t_{end\_tr}^{a\to b} = t_{end}^{oc_j^{s_b}} - t_{end}^{oc_i^{s_a}} \quad (7.2),$$

$$gap_{tr}^{a\to b} = t_{start}^{oc_j^{s_b}} - t_{end}^{oc_i^{s_a}} \quad (7.3),$$

$$rgap_{tr}^{a\to b} = t_{end}^{oc_j^{s_b}} - t_{start}^{oc_i^{s_a}} \quad (7.4).$$

Definition. The set of all transitions from a state s_a to another state s_b is defined as:

$$TS\left(s_a^{DS_1}, s_b^{DS_2}\right) = \bigcup_{\substack{i=\overline{1,p_1} \\ j=\overline{1,p_2}}} tr\left(oc_i^{s_a^{DS_1}}, oc_j^{s_b^{DS_2}}\right) \quad (8),$$

where p_1 and p_2 represents the number of occurrences of states s_a, respectively s_b, within the data streams DS_1, DS_2.

Definition. A transition tr from an occurrence $oc_i^{s_a}$ to another occurrence $oc_j^{s_b}$, having $t_{start}^{oc_j^{s_b}} - t_{start}^{oc_j^{s_a}} \geq 0$ is called a *causal transition*.

Example. Within this example we illustrate the nature of causal transitions. Figure 3 presents the possible causal transitions between occurrences of states belonging to the represented data streams DS_1 and DS_2 as arrows starting from the oldest occurrence to the newer one. In order to maintain the transitions clear we omitted on purpose the arrows starting from $oc_1^{s_b}$ and $oc_1^{s_y}$.

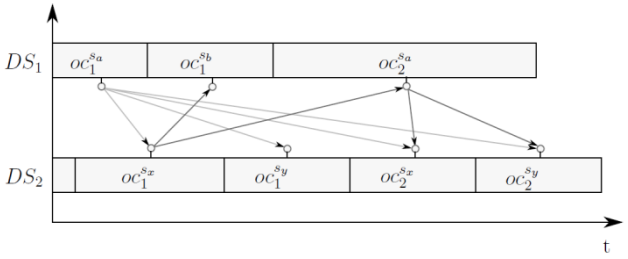

Figure 3. Causal transitions between occurrences

III. PROBLEM STATEMENT

Given a collection of heterogeneous data streams, we are interested in discovering temporal relations between states exhibited by these data streams.

Through *temporal relations* we understand pairs that can take the form $s_a \xrightarrow{p} s_b$, where p represents a temporal information strictly related to the occurrences of states it describes, in our case s_a and s_b.

Example. Given a smart office divided as within figure 4, equipped with cameras, badges and door state sensors, we are interested in discovering temporal relations between the states exhibited by the surveillance equipment in order to create a dependency model that could be used for abnormal behavior detection.

For each door within the office we have badge sensors and door state sensors. The badges record identifiers corresponding to employees and the door state sensors indicate whether the doors are opened or closed. Within each room we have cameras that can detect the objects in the room and can report the distance to them and the angle relative to the camera position.

We assume that the positioning of sensors and cameras is not known in advance.

Common employee work trajectories are represented within figure 4 and for this example we assume that they are periodically repeated.

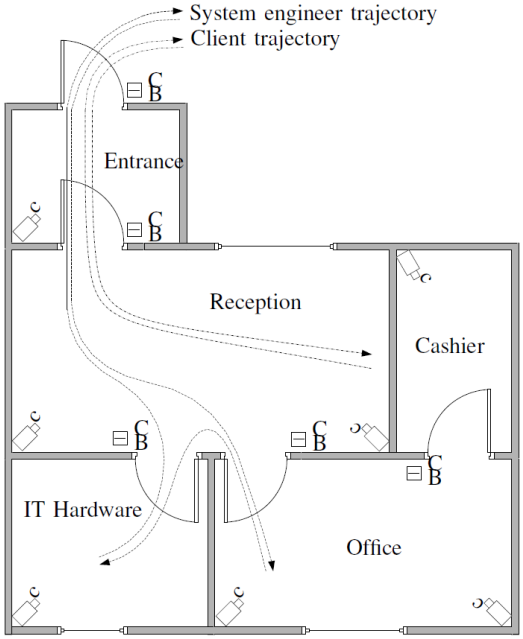

Figure 4. Smart office and common employee trajectories

The *dependency detection* problem was raised in order to extract an order out of the information supplied by surveillance equipment (a systems engineer will always follow the same trajectories for verifying the equipment and performing maintenance and thus he will trigger the same sensors in the same way each time).

The temporal dependency information extracted could be used for anomaly detection in order to reveal unusual behavior. As an example, leaving the door used to enter the IT Hardware zone opened could be signaled as an anomaly.

IV. SOLUTION APPROACH

A. *Transitions between occurrences of states*

Given n data streams DS_1, DS_2,…, DS_n we can find all transitions between all occurrences by adding a vertex into a *transitions between occurrences graph* for each detected occurrence and arcs between this vertex to and from all the other vertices in the graph.

The algorithm 1 will build a complete directed graph $TBOG(V_{TBOG}, A_{TBOG})$, that will have forward and backward arcs for each pair of occurrences. We call this graph the *transitions between occurrences graph* and we describe it using the following formulae:

$$OS = \bigcup_{\substack{j=\overline{1,n} \\ i=\overline{1,m_n}}} OC(s_i^{DS_j}) \quad (9.1),$$

$$V_{TBOG} = \{oc_k | oc_k \in OS, k = \overline{1, card(OS)}\} \quad (9.2),$$

$$A_{TBOG} = \bigcup_{\substack{k=\overline{1,n} \\ l=\overline{1,n} \\ i=\overline{1,m_k} \\ j=\overline{1,m_l} \\ i \neq j}} TS\left(s_i^{DS_k}, s_j^{DS_l}\right) \quad (9.3).$$

Algorithm 1. *Worse Case Transition Detection Algorithm*

Discover $tr\left(oc_i^{s_a}, oc_j^{s_b}\right)$ transition between all detected occurrences

REQUIRE: $DS_1, DS_2, \dots, DS_n$
transitions = {}
$OS = \bigcup_{\substack{j=\overline{1,n} \\ i=\overline{1,m_n}}} OC(s_i^{DS_j})$
for all $oc_i \in OS$ **do**
  for all $oc_j \in OS$ **do**
    if $oc_i \neq oc_j$ **then** transitions.add($tr(oc_i, oc_j)$)
  end if
  end for
end for

We called the algorithm 1 "The worst case transition detection algorithm", because it leads to a graph containing all the information coming from the data streams. It would be impossible to retain all information since we are talking about a high number of streams having a considerable throughput each.

A first optimization to the above algorithm is the allowance of causal transitions only. This optimization is made in order to reduce the number of added arcs without affecting the desired results (the dependencies between states are causal in a causal system).

Another optimization is to impose a maxgap condition while forming transitions. The condition can be expressed as follows:

Definition. Considering 2 occurrences oc_1 and oc_2 that do not overlap, having $t_{end}^{oc_1} < t_{start}^{oc_2}$, the maxgap condition to be checked while creating a transition $tr(oc_1, oc_2)$ is $t_{start}^{oc_2} - t_{end}^{oc_1} < \Delta t_{maxgap}$, and must be satisfied by all transitions that will be added to the graph, where Δt_{maxgap} is a user defined parameter.

This condition will reduce furthermore the number of transitions added to the graph and leads to a useful form of the algorithm 1 that is shown below (see algorithm 2).

With the optimized algorithm we build the *optimized transitions between occurrences graph*, $OTBOG$, which is a sub graph of $TBOG$ that contains the same vertices but has a reduced number of arcs.

B. *Transitions between states*

The $OTBOG$ graph contains all the dependency information that is relevant to us with respect to the maxgap condition and causality. But the information is organized at the level of occurrences and hence it will have a continuous growth that depends on the amount of

detected occurrences.

Occurrences represent information related to states, see the definition in section II-C, hence we can group all the vertices that represent occurrences of the same state into one vertex representing the state. By performing the grouping we lose the information that precisely locates occurrences of states in time, but we gain the ability to easily analyze the information related to transitions between states as all the transitions between the same 2 states will be represented as arcs between the same 2 vertices.

Definition. A graph that has as vertices the states exhibited by the analyzed data streams and as arcs the transitions between the occurrences of these states is called a *transitions between states graph*, $TBSG(V_{TBSG}, A_{TBSG})$. The sets V_{TBSG} and A_{TBSG} are defined in 10.

$$AS = \bigcup_{i=\overline{1,n}} S(DS_i) \quad (10.1),$$

$$V_{TBSG} = \{s_j | s_j \in AS, j = \overline{1, card(AS)}\} \quad (10.2),$$

$$A_{TBSG} = A_{OTBOG} = \bigcup_{\substack{k=\overline{1,n} \\ l=\overline{1,n} \\ i=\overline{1,m_k} \\ j=\overline{1,m_l} \\ i \neq j}} TS\left(s_i^{DS_k}, s_j^{DS_l}\right) \quad (10.3).$$

The transitions between states graph is a multigraph as we can have multiple arcs between the same vertices and one or multiple loops depending on discovered relations between occurrences of the same state type.

C. Timespan approximation

One of the problems that arise in heterogeneous environments, like smart buildings, is the occurrence of a very high number of transitions described by parameters that differ slightly. A small difference in parameter value should be allowed when searching for temporal dependencies. The variation allowed should increase with the parameters value. In our case all parameters that represent a transition are timespans.

In order to address this problem we developed a timespan approximation technique. The approximation is done using an approximation function $f(x)$ and a growth function $g(x)$.

Definition. An *approximation function* of a timespan is defined as a function $f: \mathbb{R}^+ \to \mathbb{N}$, having the property $f(\Delta_t) = \min(y)$, $y \in \mathbb{N}$, so that $\Delta_t \leq g(y)$, where g is the chosen growth function. The approximated value of the timespan Δ_t is denoted with Δ_t'.

Definition. A *growth function* is defined as strictly increasing monotone function, $g: \mathbb{N} \to \mathbb{R}^+$, whose purpose is that of defining a growth pattern.

Depending on the position of the start and end points of occurrences in a transition, sometime the computed timespans can be negative. When this happens the absolute value is approximated and the resultant value is negated.

Algorithm 2. *Optimized Transition Detection Algorithm:* Discover $tr\left(oc_i^{s_a}, oc_j^{s_b}\right)$ causal transitions between all detected occurrences that comply with the maxgap condition

REQUIRE: $DS_1, DS_2, \dots, DS_n$
transitions = {}
$OS = \bigcup_{\substack{j=\overline{1,n} \\ i=\overline{1,m_n}}} OC(s_i^{DS_j})$
for all $oc_i \in OS$ **do**
  for all $oc_j \in OS$ **do**
    if $oc_i \neq oc_j$ **then**
      if $t_{start}^{oc_j} - t_{start}^{oc_i} \geq 0$ **then**
        if $t_{end}^{oc_i} < t_{start}^{oc_j}$ **then**
          if $t_{start}^{oc_j} - t_{start}^{oc_i} < \Delta t_{maxgap}$ **then**
            transitions.add($tr(oc_i, oc_j)$)
          end if
          else
            transitions.add($tr(oc_i, oc_j)$)
        end if
      end if
    end if
  end for
end for

Example. Using a growth function $g(y) = 2^y$ and the approximation function f as it was defined earlier, the approximated values of some timespan values were computed. The values can be seen in table 1.

Table 1. Approximating timespans using $g(y) = 2^y$

Δ_t	Δ_t'	Observations
-3	-2	$y = 2, 2^2 = 4$
0	0	$y = 0, 2^0 = 1$
1	0	$y = 0, 2^0 = 1$
3	2	$y = 2, 2^2 = 4$
5	3	$y = 3, 2^3 = 8$

D. Transition similarity

The parameters that describe transitions have been defined in section II-D.

Definition. Two transitions detected between the same states are considered *similar* if and only if at least one of the descriptive parameters has the same approximated value for both transitions.

Considering the fact that the $rgap$ parameter can be computed using the other 3, while detecting similar

transitions only the combinations specified in table 2 will be taken into account, the others being redundant. In table 2, each line represents a type of similarity between 2 transitions.

Table 2. Types of similarities between transitions

$\Delta_{start\ tr}^{a\to b}$	$\Delta_{end\_tr}^{a\to b}$	$gap_{tr}^{a\to b}$	$rgap_{tr}^{a\to b}$
1			
1	1		
1		1	
1			1
1	1	1	
	1		
	1	1	
	1		1
		1	
		1	1
			1

E. Dependency detection

Algorithm 3. *Dependency detection within a window W*

REQUIRE: $transitions$ – structure created using algorithm 2
REQUIRE: W – the current window
REQUIRE: MIN, MAX – minimum and maximum number of transitions of one type in final graph

struct_dep = {}
for all $tr \in transitions$ **do**
  if $tr \in W$ **then**
    $\Delta'_{\text{start\_tr}} = aproximare(\Delta_{start\_tr})$
    $\Delta'_{\text{end\_tr}} = aproximare(\Delta_{end\_tr})$
    $\text{gap}'_{\text{tr}} = aproximare(gap_{tr})$
    $\text{rgap}'_{\text{tr}} = aproximare(rgap_{tr})$
    $struct\_dep[\Delta'_{\text{start\_tr}}].val++$
    $struct\_dep[\Delta'_{\text{start\_tr}}][\Delta'_{\text{end\_tr}}].val++$
    $struct\_dep[\Delta'_{\text{start\_tr}}][\text{gap}'_{\text{tr}}].val++$
    $struct\_dep[\Delta'_{\text{start\_tr}}][\text{rgap}'_{\text{tr}}].val++$
    $struct\_dep[\Delta'_{\text{start\_tr}}][\Delta'_{\text{end\_tr}}][\text{gap}'_{\text{tr}}].val++$
    $struct\_dep[\Delta'_{\text{end\_tr}}].val++$
    $struct\_dep[\Delta'_{\text{end\_tr}}][\text{gap}'_{\text{tr}}].val++$
    $struct\_dep[\Delta'_{\text{end\_tr}}][\text{rgap}'_{\text{tr}}].val++$
    $struct\_dep[\text{gap}'_{\text{tr}}].val++$
    $struct\_dep[\text{gap}'_{\text{tr}}][\text{rgap}'_{\text{tr}}].val++$
    $struct\_dep[\text{rgap}'_{\text{tr}}].val++$
  end if
end for
for all $items \in struct\_dep$ **do**
  if $item.val < MIN$ OR $item.val > MAX$ **then**
    struct_dep.erase(item)
  end if
end for

The dependency detection method creates a graph that has as vertices states of data streams and as arcs groups of similar transitions.

A group of similar transitions is described by a combination of parameters (see table 2), their approximated values and a weight. The weight is increased each time a new transition similar to the group is detected. The confidence in a relation increases with the weight.

The dependencies are detected using algorithm 3.

Within algorithm 3 we can observe that it relies on algorithm 2 in order to detect transitions and afterwards it uses this information to group transitions based on detected similarities.

In order to avoid the filling of the random access memory a sliding window is used, having a step equal to a quarter of the window size. The window should be significantly bigger than the smallest time intervals to be analyzed in order to avoid errors at the window borders. Algorithm 3 should be applied for each time window and the results obtained should be stored for later use.

The timespan approximation technique allows the dependency detection algorithm to group more relations than otherwise possible, with the risk of introducing some negligible errors.

In order to use the resulting dependencies the user should filter the graph using minimum and maximum allowed values for the weight. This allows the user to remove very frequent patterns that might not be relevant, as well as patterns that have been identified only once. The minimum value could be considered as a minimum allowed support for a dependency group and the maximum value as a noise filter.

The proposed architecture is based on DSCPE, Data Stream Continuous Processing Engine, [28], and is presented in figure 5. Special processing units have been implemented in order to perform the dependency detection task. The algorithms 2 and 3 have been adapted to work in a continuous environment, on an infinite amount of data coming from the analyzed data streams.

The implementation was done in C++, using elements from the STL library, C++0x standard.

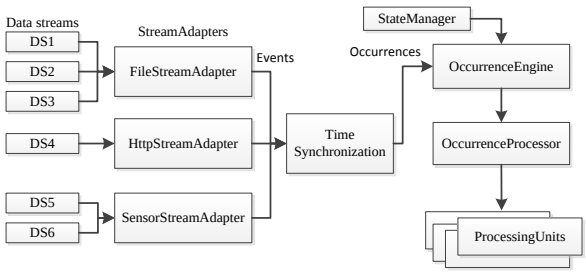

Figure 5. DSCPE Architecture

V. TESTING

A. Testbed setup

For testing the algorithm a dataset containing the recording of 4 surveillance cameras over a period of one

week was used. Two of the cameras were standard cameras and the other two were thermal cameras. The dataset has been provided by the Database team of the LIRIS research laboratory in Lyon. A research report has been presented in front of the team before this paper was published, [29].

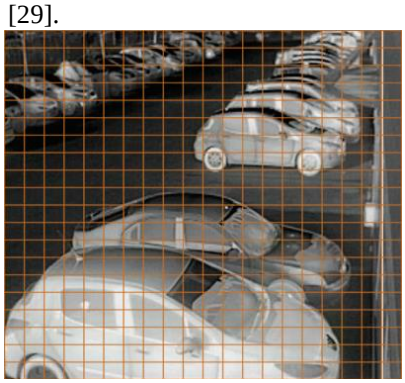

Figure 6. Division of surveillance camera into data streams

For analysis the video from each camera was divided evenly into 400 equal rectangular sections (20x20), and for each section a data stream that exhibited two states was created, *ON* when an object is within the section and *OFF* when there isn't any object within the section. In figure 6 the camera division can be observed.

Each event was timestamped using a synchronized data source. The states of each section were saved once a second in a journal file, which was used to replay the information in order to extract temporal dependencies.

The testing was performed on a system having an Intel Core i7 processor at 2.6 Ghz, 12 GB RAM, 1TB HDD and Windows 7 64 bit as operating system.

Figure 7 shows a snapshot taken using the cameras, on which we will overlay the detected dependencies.

The first operation performed on the data set was a statistical analysis, in order to show the distribution of events over 1 hour time windows. The result can be seen in figure 8. The recording begins Thursday, 01.03.2012, and ends Wednesday, 08.03.2012. The resulting distribution shows an increased activity during the week and especially during office hours.

The total number of events corresponding to the dataset is 7941835.

The following conditions were imposed while running the dependency detection process:

- Maxgap condition – maximum 5s difference between any 2 occurrences that form a transition
- Windows size – 1 hour, Step – 15 minutes
- In order for a transition to be considered valid the occurrences that form the transition must belong to the same time window.

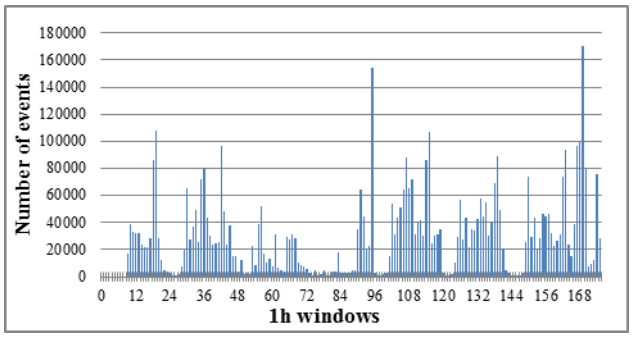

Figure 7. Number of events per hour starting at 00:00, 01.03.2012

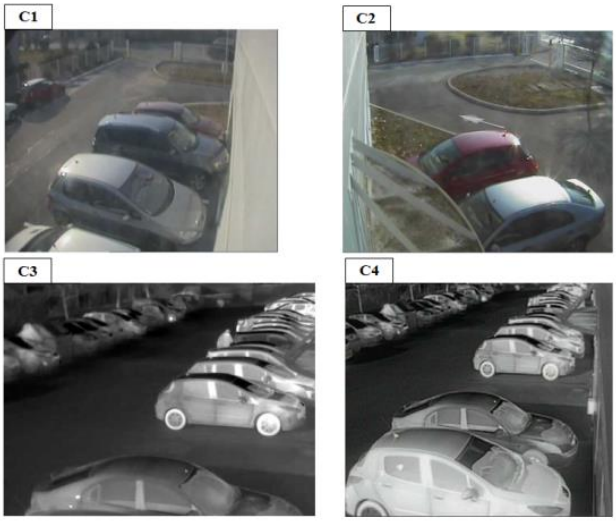

Figure 8. Snapshot from the surveillance cameras

Before creating the dependency graph using algorithm 3, we computed the number of generated transitions over 1h time windows in order to compare it with the event distribution.

Given the conditions specified above, the total number of generated transitions was 864781122. In figure 9 the distribution of the generated transitions can be observed.

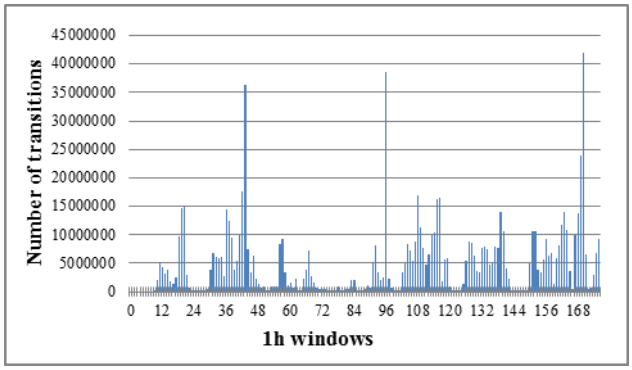

Figure 9. Number of created transitions per 1 hour window

The number of created transitions is significantly higher than the number of processed events. This high number of created transitions occupies a huge amount of

memory, thus limiting the maximum window size that can be used for analysis. We can observe a couple of peaks where the number of detected transitions exceeds 10000000. Since all operations are performed in memory these peaks can lead to a waste of resources. There has to be enough memory so that we can process peaks, even though the normal usage would be about 5 times lower.

The dependency detection operation has been performed using algorithm 3. The approximate processing time of the entire dataset is 45 minutes. Most of the time is spent on indexing operators of the *unordered_map* data structures that have been used to represent the information.

Figure 10. Intra-camera dependency information between areas in the parking lot

The first test was configured to look for temporal dependencies having a minimum weight of 20 and a maximum weight of 50. The numbers were chosen after excluding very high weights and small ones, in order to highlight the dependencies between sections in the parking lot. The results are shown within figure 10.

The green color represents dependencies for which the values of the descriptive timespans are close to 0. This means that the *ON* states for the connected sections begin almost simultaneously. Within yellow and orange we represent the dependencies with higher timespan values.

In figure 10 we can see that dependencies appear mostly on used roads and between subsequent sections of the image. The dependencies are represented only within each of the cameras.

In figure 11 we can observe how the dependencies between sections corresponding to different cameras are created, thus exposing the relationships between the areas covered by them.

The common areas are linked through green dependencies, while the others that are on the same direction are linked through yellow and orange dependencies. This means that it is possible to extract a movement model within the parking lot and other useful information.

Dependencies for which the weight exceeds the value of 100 within a time window correspond to the public roads, and they are not relevant for operations like anomaly detection, that could be performed using the dependency model.

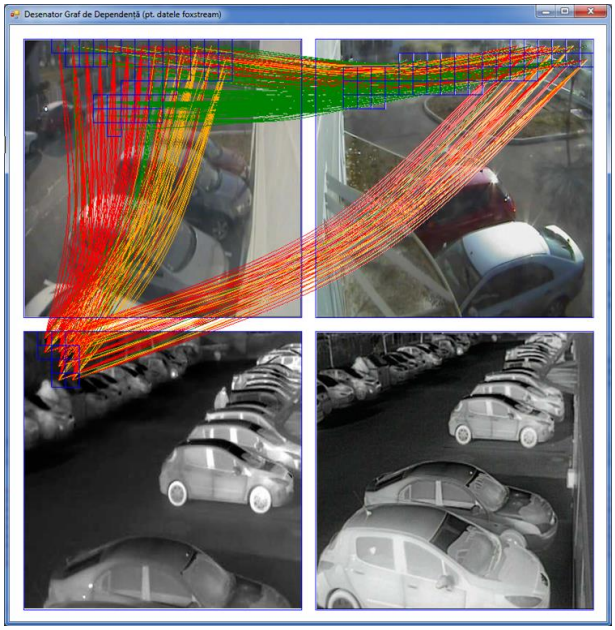

Figure 11. Inter-camera dependency information plot

VI. CONCLUSIONS

This paper is related to our dependency detection in large surveillance network project. The dependency detection method presented here is useful in systems where a large a memory is available and where the number of data streams that require analysis is also high.

The outcome of such systems represents the input for anomaly detection systems, which use the obtained dependencies to form a working model that can be used for the detection of unusual behavior.

Improvements to the current method are planned, the main directions being the optimization of the data structures used for in memory data representation. The main areas that could be improved are the continuous processing engine and the synchronization techniques used. The dependency detection solution can be improved by adding automated discovery of upper and lower bounds for filtering, in order to reduce the amount of user input.

Testing the method on data from other fields is also planned, especially on medical data and smart buildings data sets. Such data sets are available publicly or on request and the outcome can be checked against a ground truth.

The temporal pattern mining in heterogeneous data streams, that takes into account the information related to the duration of events is an emerging field. Different algorithms and methods are being tested in order to

improve learning and to produce systems that can process a high number of events and that can cope with problems posed by concept drifting and other issues. Our method provides a way of processing a high number of events as they arrive at the system and of extracting dependency information. In order to be able to perform these operations at high speeds a system with an architecture that's similar to those of continuous query systems has been implemented and tested. The results prove that the system is viable and due to this we will continue to improve the system and to add more functionality to it.

VII. ACKNOWLEDGEMENT

This paper was supported by the project "Doctoral studies in engineering sciences for developing the knowledge based society - SIDOC", contract no. POSDRU/88/1.5/S/60078, project co-funded from European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

Special thanks go to the database research team of the LIRIS laboratory, who were kind enough to provide us with a surveillance data set, on which we tested our solution.

REFERENCES

[1] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM,* vol. 26, no. 11, pp. 832-843, 1983.

[2] J. F. Allen, "Time and Time Again: The Many Ways to Represent Time," *International Journal of Intelligent Systems,* vol. 6, no. 4, pp. 341-355, 1991.

[3] C. Freksa, "Temporal Reasoning Based on Semi-Intervals," *Journal of Artificial Intelligence,* vol. 54, no. 1-2, pp. 199-227, 1992.

[4] F. Moerchen, "Temporal pattern mining in symbolic time point and time interval data," in *KDD '10 Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, 2010.

[5] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *Proceedings of 20th International Conference on VLDB*, Santiago de Chile, 1994.

[6] V. Jakkula, D. J. Cook and A. S. Crandall, "Temporal pattern discovery for anomaly detection in a smart home".

[7] T. Oates, M. D. Schmill, D. E. Gregory and P. R. Cohen, *Detecting Complex Dependencies in Categorical Data*, 1994.

[8] T. Oates, M. D. Schmill, D. Jensen and P. R. Cohen, "A Family of Algorithms for Finding Temporal Structure in Data," in *6th International Workshop on AI and Statistics*, 1997.

[9] C. Child and K. Stathis, "The Apriori Stochastic Dependency Detection (ASDD) Algorithm for Learning Stochastic Logic Rules," in *Proceedings of the 4th International Workshop on Computation Logic in Multi-agent Systems*, 2004.

[10] J. Pei, J. Han, B. Mortazavi-Asl and H. Pinto, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," in *ICDE '01 Proceedings of the 17th International Conference on Data Engineering*, Washington, DC, 2001.

[11] M. J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning,* vol. 42, no. 1 - 2, pp. 31 - 60, 2001.

[12] E. Winarko and J. F. Roddick, "ARMADA - An algorithm for discovering richer relative temporal association rules from interval-based data," *Data & Knowledge Engineering,* vol. 63, no. 1, pp. 76-90, 2007.

[13] S. Peter and F. Höppner, "Finding temporal patterns using constraints on (partial) absence, presence and duration," in *KES'10 Proceedings of the 14th international conference on Knowledge-based and intelligent information and engineering systems: Part I*, Berlin, 2010.

[14] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, 2000.

[15] J.-H. Hwang, U. Centitemel and S. Zdonik, "Fast and Reliable Stream Processing over Wide Area Networks," in *23rd International Conference on Data Engineering*, 2007.

[16] A. Metwally, A. Divyakant and E. A. Amr, "Using association rules for fraud detection in web advertising networks," in *VLDB '05 Proceedings of the 31st international conference on Very large data bases*, Oslo, 2005.

[17] C. Aggarwal, Data Streams: Models and Algorithms, Springer, 2007.

[18] S. Abiteboul, R. Agrawal and P. Bernstein, "The Lowell Database Research Self Assessment," *Communications of the ACM - Adaptive complex enterprises,* vol. 48, no. 5, pp. 111 - 118, 2005.

[19] R. Agrawal, A. Ailamaki and P. A. Bernstein, "The Claremont report on database research," *ACM SIGMOD Record,* vol. 37, no. 3, pp. 9-19, 2008.

[20] M. Franklin, A. Halevy and D. Maier, "From databases to dataspaces: a new abstraction for information management," *ACM SIGMOD Record,* vol. 34, no. 4, pp. 27 - 33, 2005.

[21] M. Franklin, A. Halevy and D. Maier, "A first tutorial on dataspaces," *Proceedings of the VLDB Endowment,* vol. 1, no. 2, pp. 1516-1517, 2008.

[22] A. Halevy, M. Franklin and D. Maier, "Principles of dataspace systems," in *PODS '06 Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, New

York, 2006.

[23] R. Avnur and J. M. Hellerstein, "Eddies: continuously adaptive query processing," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 261 - 272, 2000.

[24] S. Madden, M. Shah, J. M. Hellerstein and V. Raman, "Continuously adaptive continuous queries over streams," in *SIGMOD '02 Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, New York, 2002.

[25] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin and J. M. Hellerstein, "TelegraphCQ: continuous dataflow processing," in *SIGMOD '03 Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, New York, 2003.

[26] L. Brenna, A. Demers, J. Gehrke and M. Hong, "Cayuga: a high-performance event processing engine," in *SIGMOD '07 Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, New York, 2007.

[27] D. Gyllstrom, E. Wu and H.-j. Chae, "SASE – Complex event processing over streams," in *Proceedings of the Third Biennial Conference on Innovative Data Systems Research*, 2007.

[28] S.-S. Dragos and M.-F. Vaida, "DSCPE - A Data Stream Continuous Processing Engine," in *SICOM 2012*, Cluj-Napoca, 2012.

[29] S.-Ş. Dragoş, *Dependency Detection in Data Streams - Research Report*, Lyon, 2012.

[30] J. F. Roddick and M. Spiliopoulou, "A survey of temporal knowledge discovery paradigms and methods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 4, pp. 750 - 767, 2002.