The Journal of Systems and Software 84 (2011) 700–710

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Supporting real-time supply chain decisions based on RFID data streams

Damianos Chatziantoniou*, Katerina Pramatari, Yannis Sotiropoulos

*Department of Management Science and Technology, Athens University of Economics and Business (AUEB), Patission Ave, 104 34 Athens, Greece*

ARTICLE INFO

*Article history:*
Received 4 May 2009
Received in revised form 25 February 2010
Accepted 9 December 2010
Available online 24 December 2010

*Keywords:*
RFID data management
Data analysis
Spreadsheet 'models
Data streams
Supply chain decisions
OLAP

ABSTRACT

While RFID technology has been widely praised for its ability to streamline supply chain processes, little attention has been given to its unique data capturing characteristics to support real-time decision making. Being able to efficiently perform complex real-time analysis on top of RFID event streams is a key challenge for modern applications. This provides management with a novel data analysis mechanism to allow better, tactical, on time, well-informed decisions. The two main issues in RFID data management (RFDM) concern expressibility (how to simply and concisely express stream queries) and performance (how to efficiently evaluate stream queries). In this paper we claim that a spreadsheet-like query model, where formulation is done in a column-wise fashion, can express intuitively a large class of useful and practical RFDM queries. We propose a simple SQL extension to do that and show how these queries can be evaluated efficiently. We finally discuss a prototype called COSTES (COntinuous SpreadsheeT-likE computations), which implements our SQL extensions and evaluation algorithms. Presentation takes place within the context of two representative RFID applications, namely shelf availability and in-store sales promotions.



## 1. Introduction

The emergence of new automatic identification technologies, such as Radio Frequency Identification (RFID), is expected to revolutionize many of the supply chain operations by reducing costs, improving service levels and offering new possibilities for identifying unique product instances. The advanced data capture capabilities of RFID technology coupled with unique product identification and real-time information integrated from different data sources define a new and rich information environment that opens up new horizons for efficient management of supply chain processes and decision support.

Currently, most applications of RFID in supply chain management exploit the automation capabilities of the technology with the objective to speed-up processes and reduce costs, such as the automatic identification of incoming and outgoing goods in warehouse operations or asset tracking in closed-loop applications. However, the real potential of RFID lies in the possibility to capture new types of information in real-time and support decisions. Lee (2007) talks about moving from a substitution effect, where RFID tags are simply used to replace barcodes and automate processes, to structural effects where real-time information is exploited to create new business opportunities and experiences for the customers.

Real-time information capturing and decision support present complex technical challenges, related to managing huge streams of data coming from multiple data sources and converting them into meaningful information in a way to support decisions. Until recently, decision support systems (DSS) were based on data that were stored statically and persistently in a database, typically in a data warehouse. Complex queries and analysis were carried out upon this data to produce useful results for managers. In many applications however, it may not be possible to process queries within a database management system. These applications involve data items that arrive on-line from multiple sources in a continuous fashion. This data may or may not be stored in a database. In data streams management systems (DSMS) we usually have "continuous" queries rather than "one-time". The answer to a continuous query is produced over time, reflecting the stream data seen so far. Computing real-time analytics (potentially complex) on top of data streams is an essential component of the real-time enterprise and an essential requirement of DSMS.

In this paper we discuss the need to exploit RFID technology beyond automation in order to support real-time decisions in supply chain management. For two indicative application areas, namely shelf availability and in-store sales promotions, we define spreadsheet-like reports that improve tactical decision making based on an item and shelf-level deployment of RFID technology. We then propose an intuitive and simple SQL-based query language to express succinctly and concisely the respective reports. In section five we discuss implementation and optimization issues of the pro-

* Corresponding author. Tel.: +30 210 820 3953; fax: +30 210 820 3953.
*E-mail addresses:* damianos@aueb.gr (D. Chatziantoniou), k.pramatari@aueb.gr (K. Pramatari), yannis@aueb.gr (Y. Sotiropoulos).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.12.011

posed query language. We conclude with some critique and ideas for further research in this area.

## 2. RFID in supply chain management: requirements for real-time reports to support decisions

An RFID system consists of RFID readers with antennas, host computers and transponders or RF tags which are recognized by the readers (Piramuthu, 2007). An RFID tag is uniquely identified by a tag ID stored in its memory and can be attached to almost anything. The EPC (Electronic Product Code) standard is today the prevailing numbering scheme for specifying unique product IDs in the supply chain environment (Shih and Sun, 2005). RFID applications generate large volume of streaming data, which have to be automatically filtered, processed, grouped and transformed into meaningful information in order to be used in business applications.

RFID technology has been extensively used for a diversity of applications ranging from access control systems to airport baggage handling, livestock management systems, automated toll collection systems, theft-prevention systems, electronic payment systems, and automated production systems (Agarwal, 2001; Hou and Huang, 2006; Kelly and Erickson, 2005; Smith and Konsynski, 2001). Nevertheless, what has made this technology extremely popular nowadays is the application of RFID for the identification of consumer products and supply chain management. Applications in this field range from upstream warehouse and distribution management down to retail-outlet operations, including shelf management, promotions management and innovative consumer services (GCI, 2005; Pramatari et al., 2005). Most of these applications are characterized as "open-loop", meaning that they require the involvement of different supply chain partners in an open environment in order to be implemented, as, for example, the involvement of the supplier/manufacturer to attach an RFID tag to the product and the involvement of the distributor or retailer to monitor product movement in warehouses or stores by installing RFID readers at various locations.

This fact is probably what poses the greatest challenge for the application of RFID technology in supply chain management today, as the involved partners cannot equally share the associated costs and benefits. For suppliers, RFID, as a tag that has to be placed on their products, is often considered to be an unfortunate strategic necessity (Barua and Lee, 1997) they have to comply with in order to satisfy the plans of their big customers for increased internal efficiency. For suppliers to benefit from RFID they need to share RFID information with their partners and exploit this information in order to streamline supply chain processes and gain new market knowledge (Subramani, 2004). At the same time, for both retailers and suppliers, investments in RFID technology cannot be justified by operational gains alone and more strategic benefits need be materialized through advanced information acquisition and decision support.

Overall, and in line with Lee (2007), we see the gradual contribution of RFID in supply chain management, as an automatic product identification technology, across the following axes:

(a) the automation of existing processes, leading to time/cost savings and more efficient operations;
(b) the enablement of new or transformed business processes and innovative consumer services, such as providing consumer self check-out or product-information services;
(c) the availability of richer and more accurate information in real-time, offering the potential for advanced decision support and market knowledge acquisition.

In order to move from the level of automation and operational benefits to the level of advanced decision support we need to efficiently and effectively transform RFID data into meaningful reports, both internally within a company and in a collaborative supply chain environment where information is shared among supply chain partners (Li et al., 2006). This need is better illustrated by the following two application scenarios that have been explored in collaboration with retailers and suppliers in three different European countries, as part of the work undertaken for the SMART research project (IST-2005, FP6). These scenarios employ product item-level RFID tags, while RFID readers and antennas are placed on store shelves, so that the product movement on and off the shelf is monitored.

### 2.1. Monitoring shelf availability

This scenario refers to the possibility of monitoring the existence or not of products on supermarket shelves in order to replenish them on time. Given the negative impact that "out-of-shelf" (OOS) has on consumer attitude, sales and loyalty, retailers and suppliers have raised this issue to a top priority for their industry today and confront RFID as a possible solution to this problem. The requirements for a real-time report monitoring shelf availability would be as following:

- The report presents the remaining quantity of each product on the shelf, where a product is identified by its description at the product type level.
- The report is updated in real-time, depicting product sales off the shelf as well as shelf replenishment activities.
- A user, e.g. store employee or supplier, can view the report or get OOS notification alerts in order to make shelf replenishment decisions.
- When the last item of a product on the shelf has been sold, an OOS is reported.
- The duration of the OOS is tracked until the shelf is replenished back.

### 2.2. Promotion management

A particularly important marketing activity for fast moving consumer goods are sales promotions, which represent the majority of manufacturers' marketing budgets, amounting to 16% of their revenues (Pauwels, 2007). Despite the importance of sales promotions and the amount of revenues devoted to them, suppliers often fail to evaluate sales promotion effectiveness and when they do so, this is usually several weeks after a promotion has ended. Being able to monitor the effectiveness of in-store sales promotions in real time gives a supplier the possibility to act proactively and ensure the success of a sales promotion. In order to do so, a supplier should have access to real-time information about product sales in the store, including information about:

- The sales off-take from a specific promotional stand versus the shelf or other locations in a store. If a location is underperforming, then the supplier should probably request the store to change the location of a promotion.
- The availability of products on both the shelf and the promotion stands in order to drive replenishment decisions.
- The products that a consumer has already put in her basket when selecting a product from a promotional stand or those that she replaces as a response to a sales promotion.

The above application scenarios and respective reports are indicative examples that demonstrate the need for real-time decisions in supply chain management, exploiting the possibilities

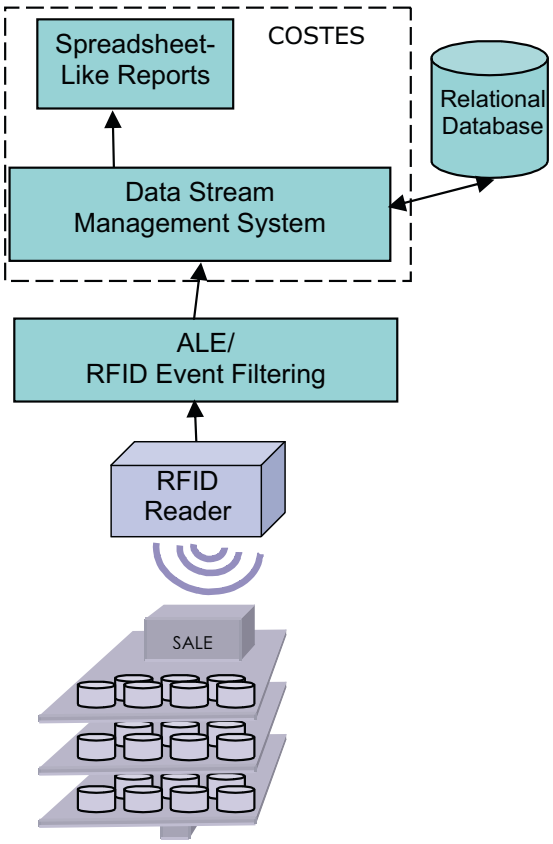



**Fig. 1.** System setting and high-level architecture.

a

| EPCProdCode | Threshold | max_Quantity | Alert |
|---|---|---|---|
| 1111 | 10 | 45 | False |
| 2222 | 12 | 7 | True |
| 3333 | 8 | 22 | False |
| 4444 | 15 | 11 | True |
| … | … | … | … |

b

| EPCProdCode | Threshold | Time_to_Repl |
|---|---|---|
| 1111 | 10 | null |
| 2222 | 12 | 3200 |
| 3333 | 8 | null |
| 4444 | 15 | 1800 |
| … | … | … |

c

| EPCProdCode | Shelf_Variance | Stand_Variance |
|---|---|---|
| 1111 | 0.61 | 0.82 |
| 6666 | 0.43 | 0.35 |
| 8888 | 0.81 | 0.84 |

d

| EPCProdCode | Threshold | avg_Quantity | Time_to_Thres |
|---|---|---|---|
| 1111 | 10 | 48 | 32400 |
| 2222 | 12 | 53 | 267988 |
| 3333 | 8 | 26 | 169366 |
| 4444 | 15 | 39 | 92102 |
| … | … | | … |

**Fig. 2.** Instance of the output of queries Q1–Q4.

offered for automatic and unique product identification through the use of RFID technology. In the following section we propose a decision support tool, utilizing a data stream management system to aggregate RFID data, addressing the aforementioned requirements for real-time decisions in the supply chain context.

## 3. Motivating examples and challenges

In order to support the aforementioned application scenarios, we have installed RFID readers in a super market monitoring the presence of products on shelves. We use the application-level-event (ALE) middleware services (ALE, 2010)) to retrieve filtered RFID data from RFID readers. We use the asynchronous subscribe mode of ALE service, where a client registers a subscription and the ALE service periodically sends aggregated events back to the client. Our system manages data streams over ALE. The setting of our system is described in Fig. 1.

Let's assume that a product may be displayed at more than one location in the store, e.g. the regular shelf position and a promotional stand. The schemata of the RFID data streams presented to our system, after filtered and aggregated by the ALE middleware, are:

```
Readings (EPCProdCode, quantity, timestamp).
Stand1 (EPCProdCode, quantity, timestamp).
...
Stand5 (EPCProdCode, quantity, timestamp).
```

Each stream tuple reports the EPC product-type code, the measured quantity of the product and the timestamp of the measurement. Streams are aggregated by the ALE service at the product-type level, i.e. the Global Trade Item Number (GTIN) occupying the first 13 digits of an EPC (usually referred to as barcode). `Readings` report products' status placed on regular shelves and `Stand1` to `Stand5` report products' status placed on stands one to five respectively. We assume that a product exists only in one location at the store's shelves and possibly at one promotional stand. While customers often misplace products, ALE middleware only reports product quantities found at their designated places (regular shelf and promotional stands). In other words, `Readings` and `Stand1` to `Stand5` streams report clean data. Based on the aforementioned application scenarios, there are several useful continuous queries one can register on top of these streams to monitor shelf replenishment and compare sales of the same product placed at different locations.

(Q1) It is important to know when a product's quantity on the shelf has reached a critical threshold and notify the store's manager to replenish it.
(Q2) Similarly, it is useful to know how long it takes to replenish the shelf (for each product), starting counting from the first occurrence of "below-the-threshold" event, in order to monitor product availability on the shelf and duration of out-of-stocks.
(Q3) During promotional periods of a compilation of products placed on a stand we are interested in measuring the effectiveness of the promotion. We can do so by comparing the sales rate of the same product from the shelf and the promotional stand.
(Q4) A more demanding report would be computing the time a product takes to get from its average quantity on the shelf to its threshold. This would allow store managers to better understand and schedule shelf replenishments and make shelf-space allocation decisions.

An instance of the output of queries Q1–Q4 is shown in Fig. 2(a)–(d) respectively.

All queries have a common pattern: the leftmost part of the output corresponds to a traditional relational expression, while the rest of the columns represent aggregates over sets of values formed over stream data. Let us consider Query Q2. The idea we want to express (in some pseudo language) is the following:

```
1:  compute table R(EPCProdCode, Threshold);
2:  add column Time_to_Repl to R;

3:  for each row r of R, define an ordered set
    S^r containing real values, initially empty;

4:  for each row r of R, set each r.Time_to_Repl = null;

5:  for each stream tuple (p, q, t) {
6:      for each r in R {
7:          if (r.EPCProdCode==p && (r.threshold > q || empty())) {
8:               push t to S^r;
9:               set r.Time_to_Repl to (last_elem - first_elem of S^r);
            }
         }
    }

10: bool empty(){
11:     if(r.Time_to_Repl is null)
12:         return false;
13:     else {
14:         set r.Time_to_Repl to (last_elem - first_elem of S^r);
15:         store r;
16:         clear S^r;
17:         r.Time_to_Repl = null;
18:         return false;
        }
    }
```

Table R is computed by a traditional relational expression. Then, for each row of R we want to define an ordered data set that contains values (timestamps) from the `Readings` stream. The timestamp of a stream data must be appended to a row's data set if a "condition $\theta$" (line 7) is satisfied – in our case if the reported EPCProdCode corresponds to the row's EPCProdCode and the reported quantity is below the row's threshold. However, when the condition does not hold, the data set must be emptied – empty( ) is a function returning always false, having as a side effect to empty the ordered set of the row. It also calculates and stores the out-of-stock duration and clears the out-of-stock variable (line 17) at the first occurrence of "above the threshold" event. Note that we have short-circuited evaluation. One could examine only the row that agrees on the EPCProdCode, but this is an optimization specific on the condition $\theta$. In other examples, a stream value may be inserted to more than one data sets.

This is the idea we want to model: start from a relational expression (the base table), use each row as a parameter to define one or more, possibly correlated, parameterized subsets of stream sources (associated sets) and extend the schema with aggregates over these. Membership to these sets should be defined declaratively, while the aggregate computations can be any user-defined aggregate function. We argue that this approach allows succinct and concise representations of many practical monitoring queries. The challenge is to have a simple query language to express and an efficient, optimizable, algorithm to evaluate such queries.

Note that each ordered data set $S^r$ is "attached" to a persistent relational value, i.e. $S^r$ can be thought as labeled by a "stable" value. This can be used to develop a relational operator to express this class of queries (Chatziantoniou and Sotiropoulos, 2008).

Based on the aforementioned real-time report requirements and the proposed system architecture, we have identified a number of requirements that a language dealing with RFID data streams should fulfil:

[R1] *Continuous queries*: The nature of the queries should be allowed to be continuous, i.e. the result can be updated as new data arrives.
[R2] *Tabular format*: The output of such a query should be in tabular format, appropriate for (subsequent) traditional database processing or input to visual tools.
[R3] *Multiple data sources*: The language should allow consolidation/aggregation of multiple data streams with different schemata into one query.
[R4] *Balancing declarative/procedural tradeoffs*: We should specify declaratively most of the query – ripping the benefits from traditional database access methods, while allowing procedural flexibility in the aggregation part.
[R5] *Correlated computations*: Many queries – especially those based on IDs – have a common pattern: the leftmost column(s) consists of the IDs (and possibly related information), while the remaining columns are defined successively based on previously defined columns.

In the following section we discuss the above requirements with respect to our proposed syntactic constructs.

## 4. COSTES queries: proposed query language

### 4.1. Background and related work

The concept of grouping variable (Chatziantoniou, 1999; Chatziantoniou and Ross, 1996) has been the primary motivation of

this work. Grouping variables have influenced at least one commercial system and SQL:1999 OLAP Amendment. Applications of them have been studied in the context of telecom applications, medical and bio-informatics, finance and others (Chatziantoniou, 2007a,b). The main idea is that one can define multiple subsets of a table using the values of the group by attributes as parameters and then aggregate over these (via a simple SQL extension). Grouping variables can be represented in relational algebra using the MD-Join operator (Akinde and Bohlen, 2001; Chatziantoniou et al., 2001). The MD-join is a combined outer-join/group by operator, clearly distinguishing the aggregate-by formulation from the aggregation process. The idea is that using the rows of a table as parameters, one can define multiple selections (using different conditions) and aggregate over these. These ideas are similar to parameterized query processing (Chaudhuri et al., 2003; Galindo-Legaria and Joshi, 2001).

In previous years, several data stream management systems (DSMS) have been developed to support stream applications (Golab and Ozsu, 2003). STREAM (Arasu et al., 2003) is a general-purpose DSMS supporting continuous queries over multiple data streams and stored relations. STREAM is based on relational semantics supporting window, relation-to-stream and relational operators. Continuous Query Language (CQL) (Arasu et al., 2006) derives from SQL:1999 and implements STREAM query semantics. CQL processing emphasizes on memory usage optimization and operator scheduling. TelegraphCQ (Chandrasekaran et al., 2003; Madden et al., 2002) extends PostgreSQL DBMS to support continuous queries over high volume and high variable data streams. Streams can be created using new DDL statements (e.g. CREATE STREAM) and continuous queries are SQL statements with an optional window clause. TelegraphCQ focuses on shared and adaptive processing of continuous queries. Aurora (Carney et al., 2002) was designed for supporting monitoring applications. It follows a workflow approach for query declaration, i.e. the tuples flow through a directed graph of processing operators. Aurora tries to maximize quality of service for streaming applications and support distributed stream processing. Besides general-purpose continuous query languages, there are several proposals for RFID data management query languages, such as Bai et al. (2006) and Bai et al. (2007). Grouping variables aimed to make certain complex data analysis queries (pivoting, correlated aggregates, hierarchical comparisons, etc.) simpler to express and easier to optimize. COSTES queries have the same aspiration, but in the context of continuous queries. These queries are very cumbersome to express in CQL or TelegraphCQ.

Although COSTES queries are presented in this paper in the context of RFID management, they can be useful in other stream application domains. In Chatziantoniou and Sotiropoulos (2007), we argued for the usefulness and generality of COSTES queries in financial stream analysis, using a wide range of query examples: computing aggregates over different window types, continuous correlated aggregation, hierarchical comparisons, etc.

The importance of correlated stream aggregates and methods to evaluate these have been presented in Gehrke and Srivastava (2001). Using the spreadsheet paradigm for On-Line Analytical Processing (Codd et al., 1993) over databases has been identified as a valuable query language (Witkowski et al., 2003, 2005). Recently, a new programming paradigm called MapReduce has attracted a lot of attention due to its simplicity and efficiency (Dean and Ghemawat, 2004). Associated sets are similar to multiple, consecutive MapReduce applications (Chatziantoniou and Tzortzakakis, 2009).

### 4.2. Definitions

**Definition 1.** Given a relational schema $B$, a relational schema of a stream source $S$ and a condition $\theta$ involving attributes of $B$ and $S$ and constants, we define the associated set of $B$, $S$ and $\theta$, denoted as Assoc($B$, $S$, $\theta$), as a collection of parameterized multisets able of storing $S$'s tuples, where the columns of $B$ serve as the parameters. Each parameterized multiset of Assoc($B$, $S$, $\theta$) is denoted as Assoc$^r$($B$, $S$, $\theta$), where $r$ is a row of $B$. $B$ is called the base-values table, $S$ the source and $\theta$ the defining condition of the associated set. Assoc$^r$($B$, $S$, $\theta$) is called the instance of the associated set with respect to $r$. □

Implementation and selection of these data structures is left to the optimizer. Note that associated sets can also be defined over traditional data sources, such as flat files and relations, as long as they present a relational interface and there is a tuple iterator over the data source. We have tried this approach to TPC-H queries over distributed data warehouses and the performance improvement over traditional query processing was up to one order of magnitude (Chatziantoniou, 2007a,b).

The schema of a COSTES query is based on repeated, consecutive definitions of associated sets and can be described by the following algorithm:

```
assume S1,S2,...,Sn stream sources;
B is the initial schema of the base-values table;
for (i=1 to n) do {
    let θi be a condition involving attributes of B and S and
        constants;
    Ai = Assoc(B,Si,θi);
    let f1(si1), f2(si2), …, fk(sik) a set of aggregate functions
        on attributes of Si;
    extend B's schema with k columns, Ai_fj(sij), j=1,2,…k
        and name it B1;
    attach a link to A^r_i at row r and column Ai_fj(sij), j=1,…,k;
    B = B1;
}
```

Note that this is just the schema of a COSTES query. How such a query is evaluated is the subject of Section 5.

**Definition 2.** Given a stream tuple $s$ of $S_j$ we say that $s$ satisfies the membership test for associated set , if $i=j$ and $\theta_i$ evaluates to true with respect to $r$ and $s$. In all other cases the return value is NULL.

### 4.3. Query language

The goal is to express a large class of practical continuous queries using some intuitive extension of SQL.

#### 4.3.1. Syntactic constructs

Formally the syntax of the proposed language is as follows:

```
select A1, A2, …, Am, {Ci.fj(s1)} i=1,…,d
from R1 , R2 , … , Rk
<where θ>
<group by A1, A2, …, Am>
extended by X1(S1), X2(S2), … , Xn(Sn)
such that θ1, θ2, ..., θn
```

where $A_1, A_2, \ldots, A_m$ attributes of $R_1, R_2, \ldots, R_k$ tables, $i$ in $\{1, 2, \ldots, n\}$, $f_j$ an available aggregate function, $S_1, S_2, \ldots, S_n$ stream sources and $s_l$ an attribute of $S_i$'s schema. The newly introduced `extended by` and `such that` clauses define the additional columns ($X_1, X_2, \ldots, X_n$, the associated sets) that will be "attached" to the relation defined by the `select...from...where...group by` clause (the initial base-values table.)

- **Select**: The `select` clause may contain one or more aggregate functions of the associated sets defined by the `extended by` clause. Since more than one aggregate functions per associated set is possible, we may have more than $m+n$ columns in the output table.
- **Extended by**: This clause is used to declare the names and the data sources of the associated sets, in a comma-separated list.
- **Such that**: This is a comma-separated list of the defining conditions of the associated sets. Condition $\theta_i$ involves attributes of the initial base-values table, constants and aggregates of associated sets $X_1, \ldots, X_{i-1}$.

Given the nature of the `such that` clause, all continuously updated associated sets are functionally dependent on some column(s) of the initial base-values table (directly or transitively).

#### 4.3.2. Examples

Assume the presence of two tables in our system, `Products(ProdCode, Threshold, ...)` keeping information on products, such as product code, threshold, location, etc. and `Promotions (PromCode, StandNumber, ProdCode, ...)`, which stores information on current and past promotions. Queries Q1–Q4 can be expressed as:

```
Q1. select ProdCode as EPCProdCode, Threshold,
          X.max(quantity) as max_Quantity,
          (X.max(quantity)< Threshold) as Alert
    from Products
    extended by X(Readings)
    such that X.EPCProdCode=ProdCode and X.size()=1
```

Query Q1 requires an associated set of size 1 to be defined for each `ProdCode` tuple of Products – to keep only the last quantity reported. We use the max aggregate function to retrieve this single value – but min, sum, etc. would be equally good. `size()` is a method that enforces $X$ to have size 1 (implementation wise, these are methods of the data structures that will implement associated sets.) `Threshold` is also required in the base table because it is used in an expression in the `select` clause.

```
Q2. select ProdCode as EPCProdCode, Threshold,
          X.diff(timestamp) as Time_to_Repl
    from Products
    extended by X(Readings)
    such that X.EPCProdCode=ProdCode and
          (X.quantity < Threshold cor empty())
```

This query requires an associated set $X$ for each `ProdCode`, which is populated by the stream's timestamp when the reported quantity drops below the threshold. However, this set has to empty whenever this condition does not hold. We use a system function called `empty()` which always returns false and as a side effect empties the corresponding set. `cor` is the short-circuited disjunction operator. The aggregate function `diff` computes the difference between the last and the first element of $X$ – forcing the optimizer to use an ordered data set for $X$.

```
Q3. select ProdCode as EPCProdCode,
          X.var(quantity) as Shelf_Variance,
          Y.var(quantity) as Stand_Variance
    from Promotions
    where PromCode=172 and StandNumber=1
    extended by X(Readings), Y(Stand1)
    such that X.EPCProdCode=ProdCode and
             X.size()=200,
             Y.EPCProdCode=ProdCode and
             Y.size()=200
```

Assume that, for those products participating in promotion 172, we want to compare their sales rate from the self and the first stand. For each such product code, we define two associated sets of maximum size 200, named $X$ and $Y$, where $X$ and $Y$ contain the reported quantity from the standard stream (`Readings`) and first stand's (`Stand1`) streams. We want to monitor the variance of those sets.

```
Q4. select ProdCode as EPCProdCode, Threshold,
          X.avg(quantity) as avg_Quantity,
          Y.diff(timestamp) as Time_to_Thres
    from Products
    extended by X(Readings), Y(Readings)
    such that X.EPCProdCode=ProdCode,
              Y.EPCProdCode=ProdCode and
             ((Y.quantity > Threshold and
              Y.quantity < X.avg(quantity) cor
              empty())
```

Finally, Query Q4 is similar to Q2, but an extra comparison between the reported quantity and the shelf's running average is

required. Consequently, we need to define for each `ProdCode` a data set *X* keeping the reported quantities and use *X*'s average to constrain membership to *Y*.

### 4.4. Discussion

By construction, the output of our queries is continuous and tabular (Requirements [R1] and [R2].) By allowing associated sets to be defined over different data sources we address requirement [R3]. Membership to associated sets is defined declaratively through the `such that` clause, which allows selection of the access methods that can be used. Aggregate computation over the defined associated sets can be anything, which adds to procedural flexibility. However, even in this case some optimization is possible, such as appropriate selection of data representation (e.g. stacks, queues, min-max heaps, etc.) of the associated sets (Requirement [R4].) Finally, the fact that aggregates of an associated set may constrain the definition of subsequent associated sets, as in Query Q4, addresses Requirement [R5].

## 5. COSTES implementation

In this section we describe the query evaluation algorithm and suggest possible optimizations. We present our implementation and provide query performance results.

### 5.1. Query evaluation and optimizations

A COSTES query can be continuously updated in a simple manner. The evaluation algorithm operates as follows:

```
1: for a tuple s of data source S {
2:   for each assoc. set X having source S {
3:     for each row r of B {
4:       if (θx is true with respect to r and s) {
5:         append s into Xr;
6:         evaluate X's aggregate functions mentioned in select clause over Xr;
         }
       }
     }
   }
```

Although the evaluation algorithm is rather simple, several optimizations are possible (for a complete set of optimizations, see Chatziantoniou and Sotiropoulos, 2008).

**Optimization 5.1**. [*Projection*] Line 5 dictates that a stream tuple *s* must be appended to the data structure representing associated set instance $X^r$. One can append to $X^r$ only those attributes of *s* needed to the computation of aggregates of *X*. For example, in query Q3, only `quantity` will be appended to associated sets *X* and *Y*.

**Optimization 5.2**. [*Filtering*] Parts of the defining condition $\theta_x$ may be relevant just to the stream tuples, i.e. the defining condition may be rewritten as $\theta_x = \theta'_x \wedge \theta_c$, where $\theta_c$ is an expression involving only attributes of *S* and constants. Some systems allow pushing simple filtering conditions to the stream source, saving iterations of the main loop at Line 3.

**Optimization 5.3**. [*Sources-sets mapping*] In some cases, we may have queries with a large number of associated sets, or multiple queries with a significant number of associated sets defined in each. Given a tuple *s* of a data source *S*, it is important to quickly locate the associated sets this tuple affects (Line 2). Besides simple data source-associated sets mapping techniques, we can also build a query index scheme (Park et al., 2007) based on the defining conditions and data stream sources, to continuously determine which associated sets must be evaluated.

**Optimization 5.4**. [*Base-table indexing*] One can analyze the defining condition of an associated set and build appropriate indexes in order to quickly locate matching rows of the base table, avoiding thus the full scan of Line 3. For example, all queries of Section 4 could benefit from the existence of a hash index on ProdCode on the associated sets, to quickly identify the matching instances to the incoming EPCProdCode.

**Optimization 5.5**. [*Data structure selection*] An associated set is a collection of multisets. The representation of associated set instances is a major issue and contributes significantly to the performance of COSTES. For example, if we want to compute the running max quantity of each product, an infinite multiset is required for each instance. Of course, such queries are never implemented as such, since a single value for each product suffices. However, this should be an optimization issue and not left to the semantics of the aggregate function. Another example is the computation of a min (or max) value of a sliding window (i.e. `size()<>0`). The most appropriate representation of the associated set instances is a circular queue with a min (or max) tracking algorithm implemented (keeps the minimum of the queue and checks at dequeuing time whether the deleted element is the minimum). A rule-based approach seems appropriate for such data structure selection.

**Optimization 5.6**. [*Scheduling*] This is a challenging issue and mostly left for future work. One can think associated sets as containers (or object instances) that are sent to different data sources in a distributed stream environment. The computation takes place locally at the stream source. However, there are many open issues that should be investigated (rate of updating results at the coordinator, distributed architecture, information to be sent, synchronization, etc.)

### 5.2. System architecture and prototype

COSTES is a C/C++ system prototype able to execute continuous queries using our proposed syntax. Fig. 3 shows the main COSTES components.

A brief description of each module follows:

- **Query input**: provides an interface where users can declare and manage COSTES queries. Users can use a textual interface to write the query or formulate it using graphical components. In addition, this module allows users to declare data source parameters (e.g. schema, type, etc.). This component generates an XML file, which contains an intermediate representation of the query.

- **Query parser**: validates the query and stores information in the metadata catalog, such as base table schema, data source names and description, associated sets attributes and aggregate functions. This catalog is used by other modules during query execution.
- **Query optimizer**: analyzes the query's syntax and accesses metadata in order to identify possible optimizations (Section 5.1). Optimization details are stored as metadata and used by the query executor.
- **Data stream source manager (DSSM)**: this component provides the functionality to handle data stream sources. Currently our system supports the following data source types: flat files, databases (ODBC), web (HTTP protocol) and XML. DSSM is a multithreaded module allowing concurrent data retrieval from several data sources. To allow synchronized data retrieval, DSSM creates a FIFO queue for each data source and provides push and pop functions for each queue.
- **Scheduler**: retrieves stream tuples from DSSM queues in a round robin fashion and forwards them to query executor. Moreover, user can prioritize stream tuple forwarding between data sources.
- **Associated sets manager**: selects, builds and manages data structures for associated sets. Currently our system supports logical and time windows which can be defined in the `such that` clause.
- **Query executor (QE)**: implements the evaluation algorithm taking into consideration the optimizations that have been made by the query optimizer module. QE allocates the initial base table, builds evaluation trees for the defining conditions and creates index structures according to the optimization parameters. It interacts with the scheduler to get stream tuples and with the associated sets manager to access the implemented data structures.

Fig. 4 shows the definition of associated set *X* of query Q1 using COSTES graphical interface, along with the query results.

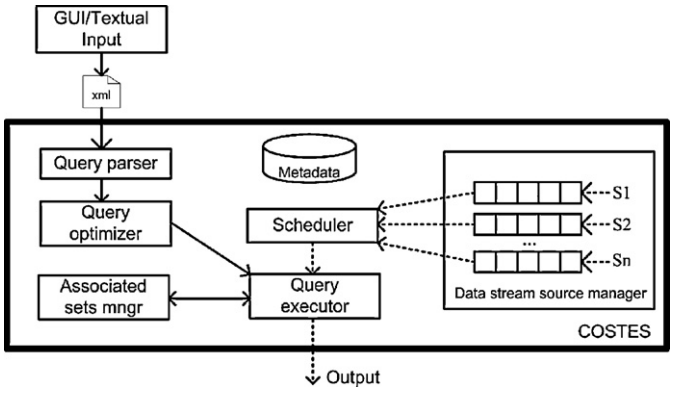


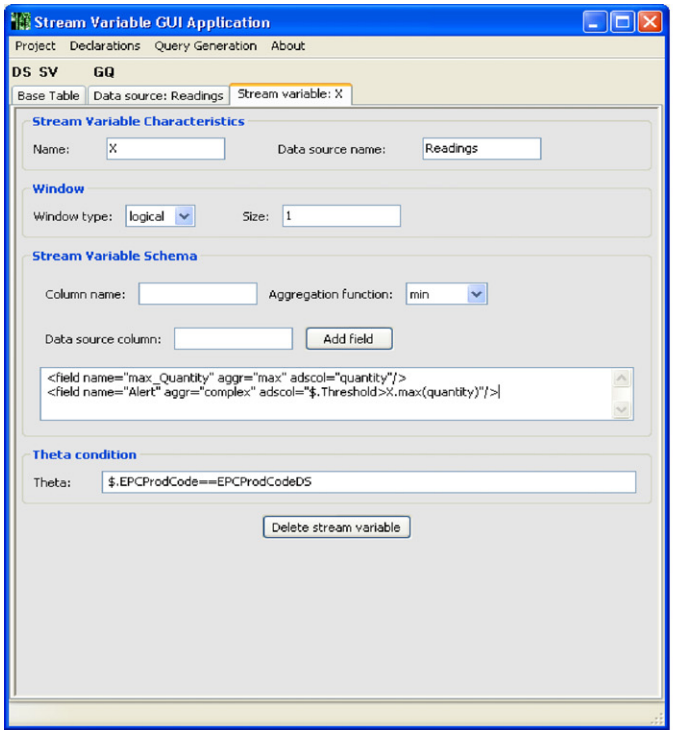

**Fig. 3.** COSTES architecture.

### 5.3. Experiments

We conducted several experiments to measure the efficiency and scalability of our system. Our tests were performed on a Pentium M 1.6 GHz with 1 GB main memory running Windows XP. In all tests we used flat files as data stream sources, since we were primarily interested in measuring the stream rate that our system could handle – the stream rate of the actual configuration was quite low. In all the experiments the base table has 5000 distinct values (i.e. 5000 distinct product codes) and the incoming stream contains pseudo-random tuples, based on actual measurements. We run three experiments, varying the size of the input stream, the size of the associated sets and the number of the data sources.

**Experiment 5.1**. In this experiment we measured the completion time of query Q1, varying the input stream size from 20,000 to 100,000 tuples having a step of 20,000, with and without Optimization 5.4. Results are shown in Fig. 5.

*Q1-Hash* line shows the performance of our system with a hash index built on the base table's attribute `ProdCode`, since the



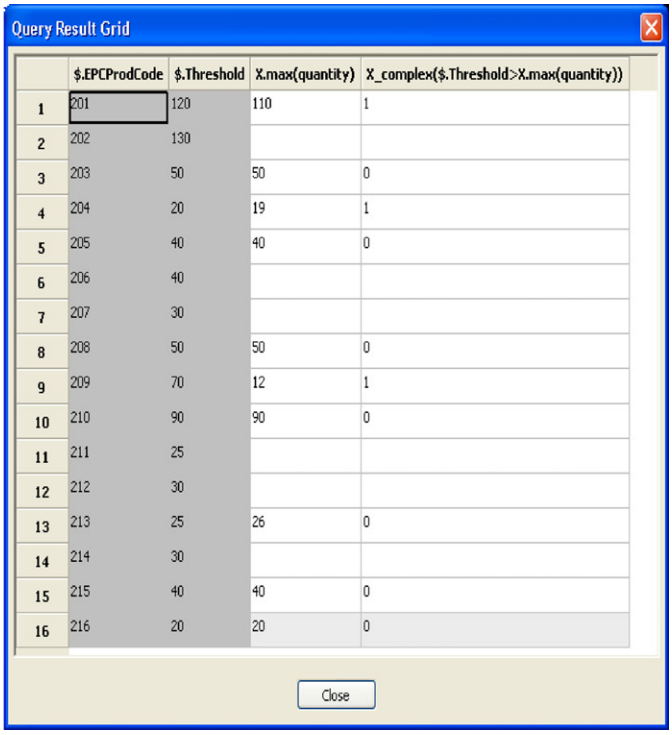




**Fig. 4.** Q1 definition and Q1 results.

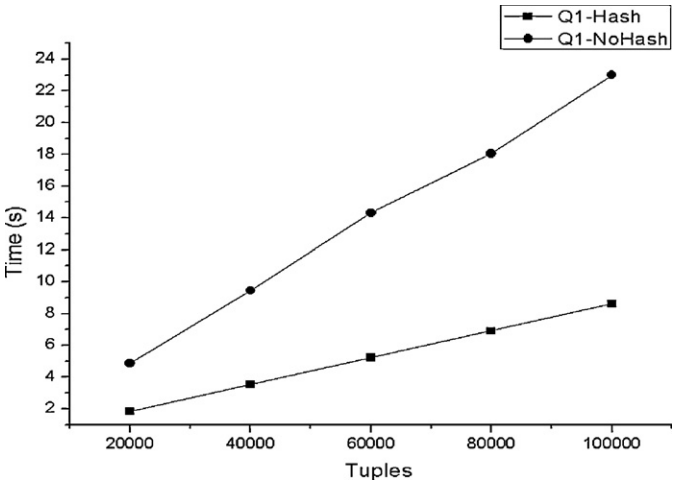



**Fig. 5.** Q1 execution time varying the number of tuples.

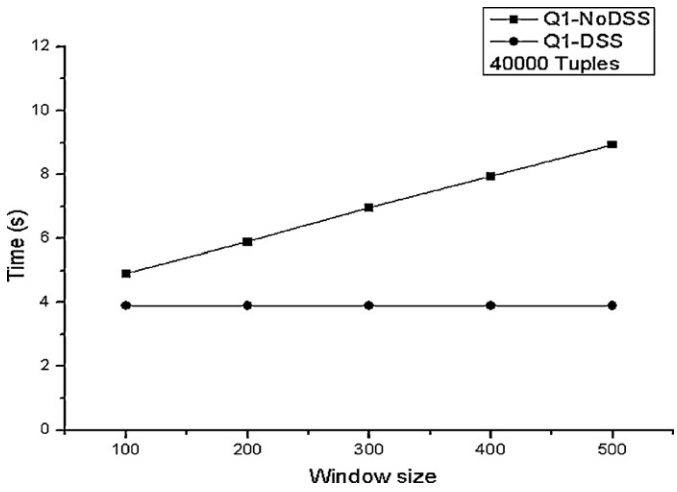



**Fig. 6.** Q1 execution time varying the window size.

optimizer has identified the (`X.EPCProdCode = ProdCode`) term in associated set *X*'s defining condition. To measure the non-indexed performance (*Q1-NoHash* line), we forced executor not to build the hash index and proceed with a naïve evaluation, i.e. a full scan of the base table for each incoming stream tuple. As expected, the former evaluation plan performs much better than the later (a factor of 2.5). Currently, COSTES system only supports single hash indexing, identified by equality predicates in conjunctive conditions. Indexes for other type of queries (e.g. range queries), multi-query optimization techniques and query indexes (Park et al., 2007) are left for future work. Another observation is that COSTES system can consume about 5000 stream tuples per second, applying unoptimized executions on similarly sized base tables. While this is a relatively good number, given that COSTES queries are specified and executed within a fully runtime environment (i.e. they are not compiled), it also shows that there is significant overhead within our system's components.

**Experiment 5.2**. In this experiment, we used a variant of query Q1, where a "true" value is reported if the product's quantity on the shelf has reached a critical threshold within a sliding window of size $n > 1$, i.e. associated set *X* has size( ) greater than 1 (the minimum value of this set is computed). We measured the performance ranging *X*'s size from 100 to 500 with a step of 100 and an input stream of 40,000 tuples, with and without Optimization 5.5. In both cases, there is a hash index built on the base table's attribute `ProdCode` (Optimization 5.4). Results are shown in Fig. 6.

*Q1-NoDSS* line shows the performance of COSTES system with no special data structure selected for the implementation of the associated set *X* (a plain queue is used). There is an increase in performance compared to Q1-Hash of Experiment 5.1 due to the management overhead of associated set *X* (is not a single value any long) and the linear search within the queue to locate the minimum element. This gap widens as the size of the queue increases from 100 to 500. *Q1-DSS* line shows the performance of COSTES system with a circular queue equipped with a min tracking algorithm for the representation of the associated set *X*. While there is an increase in performance compared to Q1-Hash of Experiment 5.1 due to the management overhead of associated set *X* (as before), the amortized cost to find the minimum element within the queue is constant. As a result, the completion time remains the same as the queue size increases.

**Experiment 5.3**. In this experiment we measured the performance of query Q3 varying the number of data sources from 1 to 100 with a step of 20. We can have a large number of stream sources in the presence of multiple stores and/or promotions. The input stream size was 40,000 tuples. Results are shown in Fig. 7.

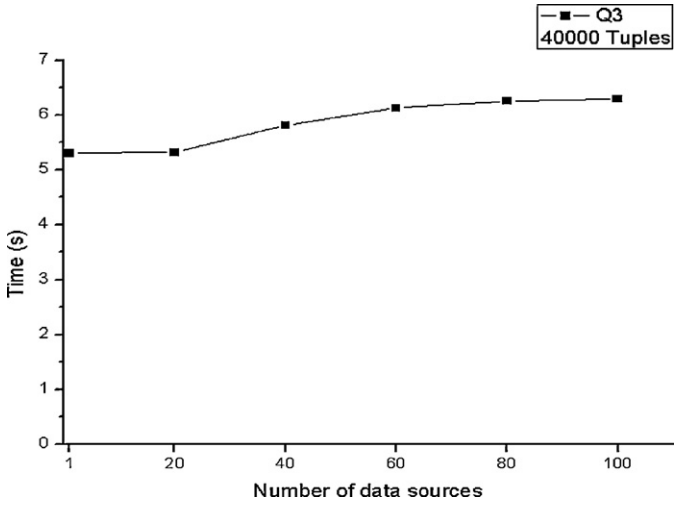



**Fig. 7.** Q3 execution time varying the number of data sources.

The incoming tuples were evenly distributed among associated sets. We assume equal arrival rate for all stream sources and one associated set defined for each data source. The optimizer builds a hash index on base table's attribute `ProdCode` (Optimization 5.4). There is a small increase in completion time due to the additional queues that the DSSM module maintains for each data source. However, as the number of queues increases, the size of each queue decreases and the data is consumed quickly by the scheduler. The construction/destruction of queue objects is the main reason for the additional performance overhead. In general, the figure shows that multiple data streams can be handled efficiently by COSTES system.

## 6. Conclusions

In this paper we described a decision support system incorporating a simple and powerful extension of SQL to express spreadsheet-like continuous computations. We argued that such an extension is particularly useful in RFID data management and presented it in the context of real-time supply chain decisions. However, this extension can be useful in other data stream application domains, such as analysis of financial streams. We briefly described how this extension could be implemented transparently to present-technology relational systems. Finally we presented a

fully functional prototype that implements this extension in a user-friendly and efficient manner.

The proposed decision support system links continuous RFID data streams with product information stored in a relational database in order to support real-time supply chain decisions. This decision support system has been incorporated in a distributed system architecture that enables information sharing between retailers and suppliers in order to enable real-time decisions in an open supply chain environment (Bardaki and Pramatari, 2007). While current RFID deployment efforts mainly deal with integrating RFID-based relational databases with existing legacy/ERP systems, the work presented in this paper moves beyond these efforts in supporting continuous queries and real-time decisions. It further contributes to transforming RFID data streams into meaningful real-time information, thus unveiling the information potential of this technology and justifying RFID investments for different supply chain partners.

As a final note, we should say that this work opens up many new areas for further research, such as assessing information sharing and information quality in this supply chain context, extending the work of Li and Lin (2006), measuring the business impact of real-time decision in the aforementioned application areas, namely the impact of monitoring shelf availability and in-store promotions, identifying information quality and management issues that are overcome or introduced through the proposed solution, as well as assessing the deployment of the proposed system in other application domains. Obviously, further research efforts as well as deployment of the proposed system in real-life settings is required in order to address these and other emerging research issues.

## Acknowledgment

This research work was partially supported by European Union ICT Grant ST-5-034957-STP and the Greek Secretariat for Research and Technology PAVE Grant 05/184.

## References

Agarwal, V., 2001. Assessing the Benefits of Auto-ID Technology in the Consumer Goods Industry. Cambridge Univ Auto-ID Centre.

Akinde, M., Bohlen, M., 2001. Generalized MD-Joins: evaluation and reduction to SQL. Databases in Telecommunications, 52–67.

Application Level Events (ALE) Standard. Available at: http://www.epcglobalinc.org/standards/ale (last accessed February 2010).

Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., Widom, J., 2003. STREAM: the Stanford stream data manager. IEEE Data Engineering Bulletin (DEBU) 26 (1), 19–26.

Arasu, A., Babu, S., Widom, J., 2006. The CQL continuous query language: semantic foundations and query execution. VLDB Journal 15 (2), 121–142.

Bai, Y., Thakkar, H., Wang, H., Luo, C., Zaniolo, C., 2006. A data stream language and system designed for power and extensibility. In: ACM Conference on Information and Knowledge Management (CIKM) , pp. 337–346.

Bai, Y., Wang, F., Liu, P., Zaniolo, C., Liu, S., 2007. RFID data processing with a data stream query language. In: IEEE International Conference on Data Engineering (ICDE) , pp. 1184–1193.

Bardaki, C., Pramatari, K., 2007. RFID-enabled supply chain collaboration services in a networked retail environment. In: Proceedings of the 20th International Bled Electronic Commerce Conference , Bled, Slovenia, June 3–6.

Barua, A., Lee, B., 1997. An economic analysis of the introduction of an electronic data interchange system. Information Systems Research 8 (4), 398–422.

Carney, D., Cetintemel, U., Cherniack, C., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, B.S., 2002. Monitoring streams – a new class of data management applications. In: 28th International Conference on Very Large Data Bases , pp. 215–226.

Chandrasekaran, S., Cooper, W., Deshpande, A., Franklin, J.M., Hellerstein, M.J., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, A.M., 2003. TelegraphCQ: continuous dataflow processing for an uncertain world. In: Conference on Innovative Data Systems Research (CIDR).

Chatziantoniou, D., Ross, K., 1996. Querying multiple features of groups in relational databases. In: 22nd International Conference on Very Large Databases (VLDB) , pp. 295–306.

Chatziantoniou, D., 1999. Evaluation of ad hoc OLAP: in-place computation. In: ACM/IEEE International Conference on Scientific and Statistical Database Management (SSDBM) , pp. 34–43.

Chatziantoniou, D., Akinde, M., Johnson, T., Kim, S., 2001. The MD-Join: an operator for complex OLAP. In: IEEE International Conference on Data Engineering , pp. 524–533.

Chatziantoniou, D., Sotiropoulos, Y., 2007. Stream variables: a quick but not dirty SQL extension for continuous queries. In: IEEE International Conference on Data Engineering (ICDE) Workshops , Ambient Intelligence, Media and Sensing (AIMS), pp. 19–28.

Chatziantoniou, D., 2007a. Associated Sets Applied on TPC-H Queries: A Performance Study. Aster Data Inc., Redwood City, CA.

Chatziantoniou, D., 2007b. Using grouping variables to express complex decision support queries. Data and Knowledge Engineering (DKE) 61 (1), 114–136.

Chatziantoniou, D., Sotiropoulos, Y., 2008. Relational management of data stream windows. ELTRUN Working Paper Series, www.eltrun.gr.

Chatziantoniou, D., Tzortzakakis, E., 2009. ASSET queries: a declarative alternative to MapReduce. ACM SIGMOD Record 38 (2), 35–42.

Chaudhuri, S., Kaushik, R., Naughton, J., 2003. On relational support for XML publishing: beyond sorting and tagging. In: ACM SIGMOD, Conference on Management of Data , pp. 611–622.

Codd, E.F., Codd, S.B., Salley, C.T., 1993. Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate. Arbor Software.

Dean, J., Ghemawat, S., 2004. MapReduce: simplified data processing on large clusters. Operating Systems Design and Implementation (OSDI), 137–150.

Galindo-Legaria, C., Joshi, M., 2001. Orthogonal optimization of subqueries and aggregation. In: ACM SIGMOD, Conference on Management of Data , pp. 571–581.

GCI, 2005. EPC: A Shared Vision for Transforming Business Processes. Global Commerce Initiative (GCI) in Association with IBM.

Gehrke, K.F., Srivastava, D., 2001. On computing correlated aggregates over continual data streams. In: ACM SIGMOD, Conference on Management of Data.

Golab, L., Ozsu, M.T., 2003. Issues in data stream management. ACM SIGMOD Record 32 (2), 5–14.

Hou, J.L., Huang, C.H., 2006. Quantitative performance evaluation of RFID applications in the supply chain of the printing industry. Industrial Management & Data Systems 106 (1), 96–120.

Kelly, E.P., Erickson, S.G., 2005. RFID tags: commercial applications vs. privacy rights. Industrial Management & Data Systems 105 (6), 703–713.

Lee, H., 2007. Peering through a glass darkly. International Commerce Review 7 (1), 60–78.

Li, J., Sikora, R., Shaw, M.J., Tan, G.W., 2006. A strategic analysis of inter organizational information sharing. Decision Support Systems 42 (1), 251–266.

Li, S., Lin, B., 2006. Accessing information sharing and information quality in supply chain management. Decision Support Systems 42 (3), 1641–1656.

Madden, S., Shah, M.A., Hellerstein, J.M., Raman, V., 2002. Continuously adaptive continuous queries over streams. In: ACM SIGMOD, Conference on Management of Data , pp. 49–60.

Park, J., Hong, B., Ban, C., 2007. A continuous query index for processing queries on RFID data stream. In: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) , pp. 138–145.

Pauwels, K., 2007. How retailer and competitor decisions drive the long-term effectiveness of manufacturer promotions for fast moving consumer goods. Journal of Retailing 83 (3), 297–308.

Piramuthu, S., 2007. Protocols for RFID tag/reader authentication. Decision Support Systems 43, 897–914.

Pramatari, K., Doukidis, G., Kourouthanassis, P., 2005. Towards 'smarter' supply and demand-chain collaboration practices enabled by RFID technology. In: Vervest, P., Van Heck, E., Preiss, K., Pau, L.F. (Eds.), Smart Business Networks. Springer Verlag, ISBN 3-540-22840-3.

Shih, D.H., Sun, P.L., 2005. Securing industry-wide EPCglobal network with WS-security. Industrial Management & Data Systems 105 (7), 972–996.

Smith, H., Konsynski, B., 2001. Developments in practice X: radio frequency identification (RFID) – an Internet for physical objects. Communications of the AIS 12, 301–311.

Subramani, M., 2004. How do suppliers benefit from information technology use in supply chain relationships? MIS Quarterly 28 (1), 45–73.

Witkowski, A., Bellamkonda, S., Bozkaya, T., Dorman, G., Folkert, N., Gupta, A., Sheng, L., Subramanian, S., 2003. Spreadsheets in RDBMS for OLAP. In: ACM SIGMOD Conference , pp. 52–63.

Witkowski, A., Bellamkonda, S., Bozkaya, T., Naimat, A., Sheng, L., Subramanian, S., Waingold, A., 2005. Query by excel. In: International Conference on Very Large Databases (VLDB) , pp. 1204–1215.

**Damianos Chatziantoniou** received his B.Sc. in Applied Mathematics from the University of Athens (Greece) and continued his studies in Computer Science at Courant Institute of Mathematical Sciences of New York University (M.S., 1993) and Columbia University (Ph.D., 1997). His research interests include Data Warehousing, OLAP, Decision Support Systems, Query Processing and Data Streams. He has published more than 25 articles at VLDB, ICDE, EDBT, ACM SIGKDD, ACM SIGMOD, Journal of Information Systems, Journal of Data and Knowledge Engineering and elsewhere. His research work has influenced Microsoft SQL Server (Query Processor), Oracle 8i and 9i (Analytic Functions for OLAP), and ANSI SQL Standard (OLAP Amendment). He is currently an Assistant Professor at Athens University of Economics and Business (AUEB) – Department of Management Science and Technology and a Senior Research Consultant in a start up based in Redwood City, CA. Prior to AUEB, Damianos has taught at New York University and Columbia University as an Adjunct Professor and at Stevens Institute of Technology as a Tenure-track Assistant Professor. He has collaborated with AT&T Research and Columbia Medical Informatics

Department. Besides academia, Damianos has been involved in a couple of technology start-up companies, one based in New York City (Panakea Software Inc., OLAP tools & consulting, founder) and one based in Athens (VoiceWeb, speech & telecom applications, co-founder).

**Katerina Pramatari** is an Assistant Professor at the Department of Management Science and Technology of the Athens University of Economics and Business (AUEB). She holds a B.Sc. in Informatics and M.Sc. in Information Systems from AUEB, and a Ph.D. in Information Systems and Supply Chain Management also from AUEB. She has won both business and academic distinctions and has been granted eight state and school scholarships. Her research and teaching areas are supply and demand chain collaboration, traceability and RFID, e-procurement, e-business integration and electronic services. She has published several papers in scientific journals including the Information Systems Journal, the Journal of Information Technology, The European Journal of O.R., Computers and O.R., Supply Chain Management: An International Journal, International Journal of Information Management and others.

**Yannis Sotiropoulos** is a Ph.D. candidate at the Athens University of Economics and Business (AUEB), Department of Management Science and Technology. He holds a M.Sc. from University of Piraeus in Network-based Systems and a B.Sc. from the Department of Technology Education and Digital Systems from University of Piraeus. Currently he is a research officer in the Center of Studies on Business Intelligence and Databases (CUBE) group of the ELTRUN Research Center at AUEB.