Data Workflow - A Workflow Model for Continuous Data Processing

Andreas Wombacher
Database Group,
University of Twente,
Enschede, The Netherlands
Email: a.wombacher@utwente.nl

***Abstract*—Online data or streaming data are getting more and more important for enterprise information systems, e.g. by integrating sensor data and workflows. The continuous flow of data provided e.g. by sensors requires new workflow models addressing the data perspective of these applications, since continuous data is potentially infinite while business process instances are always finite.**

In this paper a formal workflow model is proposed with data driven coordination and explicating properties of the continuous data processing. These properties can be used to optimize data workflows, i.e., reducing the computational power for processing the workflows in an engine by reusing intermediate processing results in several workflows.

I. Introduction

Online data, i.e., streaming data, are becoming more and more important in enterprise applications. Applications facilitate the fact that data are acquired immediately electronically and then often made accessible to other enterprise information systems. An example is the usage of sensor data (like e.g. GPS coordinates) providing context information about a user. This information is used in all kinds of context aware information systems, like e.g. location based services.

Sensor information (i.e., context information) is continuously acquired and published. Information systems have to continuously process this information. That is, after a specified number of information has been accumulated, the available information is processed. Workflows where the availability of data coordinates (i.e., controls) the processing in the workflow are called data driven workflows. Classical business workflows are coordinated by interactions with humans or other information systems (called control flow driven) and terminate after a case is complete. Processing of continuous sensor data (i.e., streaming data), however, does not terminate without user interaction, since a stream is per definition infinite.

An example of a data driven workflow is an online navigation system providing additional location based services to the driver of a car, like the availability of gas stations, rest rooms, weather forecasts, or accommodations. All of the afore mentioned location based services rely on the GPS coordinates. The GPS coordinates provide the context for the various location based services. Each location based service can be represented by a single data driven workflow. All location based workflows have to acquire the GPS signal and potentially pre-process the signal. This acquisition and pre-processing is shared between all workflows. Assume all workflows run in parallel on the same hardware. Since the processing is continuous, each workflow related to a location based service performs the GPS signal acquisition and pre-processing in parallel. The aim is to identify fragments of workflows performing the same operations on the same data and sharing them between the workflows instead of calculating them several times in parallel. This reduces the workload on the workflow engine.

In this paper, a data driven workflow model is proposed called **data workflow** supporting sharing of intermediate processing results between different data workflows. First an overview of the basic ideas is presented (Sect III), followed by the syntactic (Sect IV) and semantic (Sect V and VI) definition. Furthermore, the optimization of data workflows to reduce the required computational resources is illustrated on behalf of an example (Sect VII). A prototypical open source implementation is briefly presented in Sect VIII.

Use Case: Thomas is a sales person of a company equipped with a mobile phone and a car with GPS coordinates. To avoid road blockage due to bad weather conditions [1] Thomas uses a location based service which combines the 5 minutes average of the car's GPS coordinates with a precipitation radar available online [2]. If in the vicinity of the car's GPS coordinates there is heavy precipitation, the application sends an SMS to Thomas to warn him. This service is continuously running, since Thomas is traveling a lot (see upper part Fig 1).

To support Thomas in his heavy schedule, Anna, his secretary, has access to Thomas' GPS coordinates. Knowing Thomas' agenda and the 15 minutes average of the car's GPS coordinates, an application can estimate the potential arrival time at the next appointment. If the application detects a delay, Anna receives a warning on her computer notifying her of the potential delay and triggers her to reschedule his appointment if necessary (see lower part Fig 1).

[1]In the Netherlands, this winter highways got closed due to unusual amounts of snow.

[2]http://www.buienradar.nl/

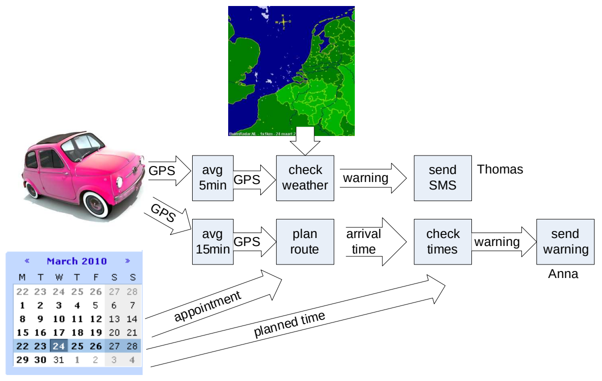

Figure 1. Location based Use Case of Continuous Data Processing

II. Related work

Continuous data processing and workflows have been investigated in different domains. Business workflows are quite different from continuous data processing [1]. The closest workflow model are scientific workflow models [2]. Dataflow process networks [3], [4] are stream based data processing approaches forming the basis for many scientific workflow systems [2], like e.g. Kepler [5], [6] or Taverna [7], [8]. A dataflow process network specifies that each stream element is read at most once from an input stream, the read data is transformed, and new data is produced. The control flow of a data flow process network is controlled by the consuming activity via a data pull. The activities in the scientific workflow performing the actual data transformation are not restricted. In this generic model information required to perform an activity is buffered by an activity itself. Keeping so much information implicit in an activity makes data sharing between different workflows difficult, since workflow optimization does not have access to implicit buffer mechanisms.

Stream extensions of scientific workflows have been proposed (e.g. [9], [10]) and implemented. However, the proposed approaches stay withing classical scientific workflow specifications and do not explicate the internal buffers used by processing steps. However, this information is essential for sharing processed data between different workflows.

Previous research focusing on reuse or sharing of workflows addresses reuses of workflow specifications (e.g. [11]) instead of sharing of data continuously processed in several workflows as addressed in this paper.

In stream data management approaches, like e.g. Global Sensor Network [12], [13], STREAM [14], [15], TelegraphCQ [16], [17], or Borealis [18], [19], data are pushed through the system. All approaches provide a sliding window mechanism supporting different window types, specifying a buffer, and sliding mechanisms, specifying the coordination on processing the data. However, the data processing is limited to functionality provided by the query language.

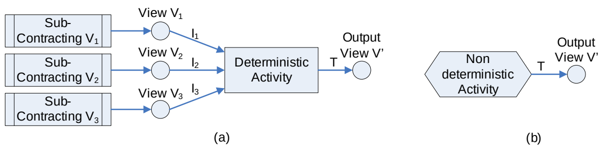

Figure 2. Processing step schema

Although SQL:2003 standard [20] provides a standard extension mechanism called *stored procedures*, this extension may not be available in stream processing query languages. In addition, using stored procedures is costly and diminishes the optimization possibilities of the data management system. However, explicating sliding windowing mechanisms provides information on data driven control flows and will be applied in the proposed approach. Stream data management produces a stream of data which can be exported in a relational database and queried there. However, the query mechanism and language differ from the stream data management language.

III. Data Workflow Approach

The main idea is to reduce computational resources by sharing information between different process instances. To achieve this goal a workflow model is required which is focused on data aspects. The proposed workflow schema is based on relational schemes and activities applied on these schemes. A relational schema consists of (i) the name of the relation and (ii) a set of attributes associated with their attribute domains [21]. An instance of a relation schema is a relation which contains a set of tuples. The standard operations for relations are projection π, selection σ, and cross-product $\times$. Further, tuple operations are defined on relations such as insertion $R \cup \langle a_1, \ldots, a_n \rangle$ and removal $R - \langle a_1, \ldots, a_n \rangle$ of a tuple $\langle a_1, \ldots, a_n \rangle$ [21].

A relation with the same name as a relation schema adheres to the schema and all tuples contained in the relation have values addressable via the corresponding attributes adhering to the attribute domain.

The minimal computational step of the workflow schema is called a processing step. A processing step is based on a potentially empty set of input relations called input views, which are the result of a network of processing steps abstracted in an subcontracting activity. Further, a processing step has an activity and a single output relation called output view. Views are graphically represented as circles and activities as rectangles. Fig 2a) depicts a processing step with three input views, while Fig 2b) depicts a processing step with no input views. Activities can be classified in three categories: sub-contracting activities representing a network of processing steps not further represented in a workflow schema, deterministic and non-deterministic activities [3] representing activities without and with an internal state. An example of a non-deterministic activity is a GPS sensor like in the use case. The sensor produces streaming data added to the output relation V', while the next GPS value added to the view can not be determined based on other input data. Examples of deterministic activities are average calculations, summations, union, join, and all kinds of aggregation functions. A complete formal definition of the workflow schema is given in Sect IV. To limit the information used from the input view an interval predicate per input view (in Fig 2a) interval predicates I_1, I_2, I_3) is defined. The interval predicate is used as a selection on the input view resulting in a set of selected tuples represented in Fig 3 as Buffers B_1, B_2 and B_3, which are then actually used by the activity to create new tuples in the output view. An activity is executed when a certain trigger predicate T is fulfilled (s. Fig. 2). A trigger predicate T is defined per activity and is valuated on the union of the buffers (s. Fig. 3).

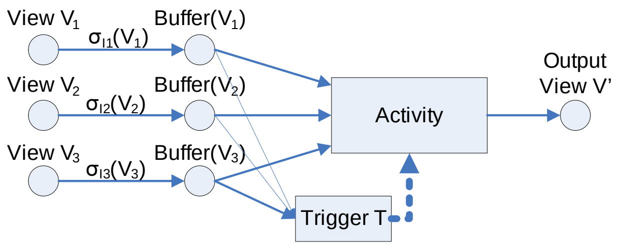

Figure 3. Schematic processing step execution

A detailed discussion of the execution is given in Sect V and VI.

The proposed approach is based on the following assumptions derived from continuous data processing applications:

- The relational schema of the involved relations does not change.
- Each relational schema has an unique ID attribute to identify the tuple within the relation.
- Tuples can only be added to a relation. No tuples can be removed. Updates are treated special.
- Executing an activity is an atomic and isolated operation on relations.
- The set of possible relations and tuples in a relation is not predictable and changes over time (open world assumption).

The explication of buffer specifications (i.e. interval predicates) and coordination mechanisms (i.e. trigger predicates) enables data sharing between different workflows to reduce the required computational resources. In Sect VII an example based on the use case is introduced.

[3]The terms deterministic and non-deterministic activities are not intuitive, since non-deterministic activities are deterministic if the internal state is known. The terms are used nevertheless since they are used in the community [4].

The design principles applied in the data workflow approach are: data flow dominates control flow, data driven coordination, and modular specifications.

In workflows a control flow describes how the execution of activities is coordinated. Data flow describes which data are exchanged between which activities. The aim of the data workflow approach is to process data and therefore data processing is the main aim of the modelling.

The coordination of a workflow is based on the data so far processed in this workflow. Thus, the coordination of an activity depends only on data produced at the last execution of the activity and the data available as input for the activity.

This coordination is based on data local to an activity and therefore supports a modular workflow specification following chained workflows [22] as a type of a distributed workflow. A modular specification allows to combine workflows at any activity result and facilitates tuple based data alignment [23].

IV. Data Workflow Schema

A workflow schema describes the structure of the workflow. The data workflow is a bi-partied graph consisting of relations, activities, and flow relations between them. Associated to input flow relations are interval predicates. Activities are associated with a trigger predicate and a classification of the activity into deterministic, non-deterministic, and chained. The data workflow schema is formally defined in Def 1.

Definition 1 (workflow schema): Let $\widehat{\mathcal{R}}$ be the universe of relations and $\widehat{\mathcal{A}}$ be the universe of activities. A data workflow schema is a tuple $W = (\mathcal{R}, \mathcal{A}, \mathcal{F}, \tau, \iota, \kappa)$ such that

- $\mathcal{R} \subseteq \widehat{\mathcal{R}}$ is a set of *relations,*
- $\mathcal{A} \subseteq \widehat{\mathcal{A}}$ is a set of *activities,*
- $\bullet\mathcal{F} \subseteq (\mathcal{R} \times \mathcal{A})$ is a set of directed arcs, called *input flow relations,*
- $\mathcal{F}\bullet \subseteq (\mathcal{A} \times \mathcal{R})$ with $|\mathcal{F}\bullet| = 1$ is a set of directed arcs, called *output flow relations,*
- $\mathcal{F} = \bullet\mathcal{F} \cup \mathcal{F}\bullet$ is called *flow relations,*
- $\tau : \mathcal{A} \rightarrow \mathcal{T}$ assigns a *trigger predicate* to an activity,
- $\iota : \bullet\mathcal{F} \rightarrow \mathcal{I}$ assigns an *interval predicate* to an input flow relation, and
- $\kappa : \mathcal{A} \rightarrow \{det, \overline{det}, c\}$ classifies an activity as deterministic, non-deterministic, or chained.

A data workflow is **well-formed** if all activities have either no input relation or have input relations, which are output relations of other activities. □

A data workflow is graphically represented as a graph, where circles represent relations, rectangles represent deterministic activities, hexagons represent non-deterministic activities, and boxes represent chained activities. Arrows between graphical elements illustrate flow relations. Interval and trigger predicates are associated with input and output flow relations.

Parts of the data workflows described in the use case (Sect I) of Thomas and Anne are depicted in Fig 4. Chaining is illustrated in Fig 4 b) using activity $check\ weather$ as a chain to the deterministic activity of the same name in Fig 4a).

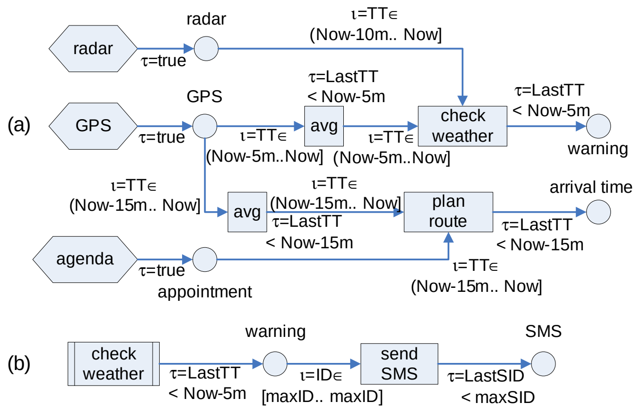

Figure 4. Example data workflows

A. Interval Predicates

An interval predicate specifies a subset of information provided by an input relation used by an activity. An interval predicate is a conjunction of constraints on attributes of the input relation, where constraints are expressed as intervals. Due to the limitation of interval predicates e.g. subsumption of two predicates can efficiently be decided. Subsumption is an important operation for optimizing data workflows as illustrated in Sect VII.

An interval is ”a set containing all points (or all real numbers) between two given endpoints” [24] (see Def 2).

Definition 2 (interval): An interval Int consists of round brackets ’(’ and ’)’ and/or square brackets ’[’ and ’]’ indicating open/closed intervals. Lower and upper interval endpoints are numerical expressions with variables $V = \{Now, maxID\}$. □

Endpoints are specified as absolute values, or relative via a numerical expression (see Def 3 to a variable Now representing the current point in time and $maxID$ representing the maximum ID of a tuple in an input relation.

Definition 3 (numerical expression): A numerical expression over a set V of variables is given as: (i) all numbers are numerical expressions, (ii) all variables in V are numerical expressions, and (iii) for numerical expressions N_1 and N_2 the expression $N_1 \theta N_2$ with $\theta \in \{+, -, *\}$ is a numerical expression. □

In an interval predicate (see Def 4) a subset of tuples in an input relation is defined by an interval (see Def 2) constraining the transaction time, i.e., the time when a tuple has been inserted to an input relation, or the ID of a tuple in the input relation.

Definition 4 (interval predicate): An interval predicate is given as: (i) constants $true$ and $false$ are interval predicates, (ii) the time when a tuple has been created (transaction time TT) and the unique ID of a tuple (ID) related to an interval, i.e. $TT \in Int$ and $ID \in Int$, are interval predicates, and (iii) for interval predicates P_1 and P_2 the conjunction $P_1 \wedge P_2$ is an interval predicate.

The set of all interval predicates is represented as $\mathcal{I}$. □

Interval predicates are limited to conjunctions since disjunctions are expressible as Union activities and therefore available for workflow optimization. Since union and negation would allow to represent disjunctions negation is also not considered.

Typical **examples** of interval predicates are fixed windows, time or count based sliding windows as used in stream data management [17]. A fixed window is an absolute interval predicate specifying historical data, like e.g. the interval predicate $TT \in [01.11.08..01.12.08)$ specifies data of month November in 2008. The interval predicate $TT \in (Now - 5m..Now]$ is a time based sliding window specifying data inserted to the input relation in the last 5 minutes (see Fig 4a). A count based sliding window specifying the last tuple inserted is given by the interval predicate $ID \in [maxID..maxID]$ (see Fig 4b). In case of interval predicates specifying sliding windows, the insertion of a new tuple in the relation changes the tuples selected by the interval predicate.

B. Trigger predicate

A trigger predicate represents the data driven control flow of the data workflow. If the trigger predicate is true then the activity can be executed. In general there are time and tuple related trigger predicates. Time related triggers are defined as an inequality of the time ($LastTT$) the activity has been executed last and the current time (Now). Tuple related triggers are expressed as an inequality of the variable SID, i.e. the direct product [4] of IDs inserted last into each input relation, and a corresponding direct product of IDs ($LastSID$) triggering the last execution of the activity. [5] Trigger predicates are defined in Def 5.

Definition 5 (trigger predicate): A **time based trigger** predicate is an inequality of variable $LastTT$ on the left hand side, a comparison operator $(<, \leq, =)$, and a numerical expression with variable $V = \{Now\}$ on the right hand side. The set of all time based trigger predicates is $\mathcal{T}_{TT}$.

A **tuple based trigger** predicate is an inequality of the variable $LastSID$ on the left hand side, a comparison operator $(<, \leq, =)$, and a numerical expression with variable

[4]A direct product results in a tuple of the IDs.

[5]Definitions of variable semantics are provided in Sect V.

$V = \{maxSID\}$ on the right hand side. The set of all tuple based trigger predicates is $\mathcal{T}_{\mathcal{SID}}$.

A trigger predicate is either a time or a tuple based trigger predicate. The set of all trigger predicates is
$\mathcal{T} = \mathcal{T}_{\mathcal{SID}} \cup \mathcal{T}_{\mathcal{TT}}$. □

Trigger predicates support inequalities to provide higher flexibility in formulating trigger predicates.

An **example** of a time based trigger predicate is $LastTT < Now - 5m$ specifying that an activity is executed every 5 minutes (see Fig 4a). An example of a tuple based trigger predicate is $LastSID < maxSID$ specifying that an activity is executed after at least one tuple has been inserted in one of the input relations selected by an interval predicate (see Fig 4b).

C. Workflow Operations

The data workflow of the use case (see Sect I) is based on a GPS sensor and information from additional information systems represented in Fig 4a). The non-deterministic activity GPS with no input relation and a trigger predicate $true$ indicates that any time a tuple might be inserted in output relation GPS. Relation GPS is an input relation to activity avg, which is applied to all tuples observed in the last 5 minutes (interval predicate $TT \in (Now - 5m..Now]$) executed every 5 minutes (trigger predicate $LastTT < Now - 5m$).

The data workflow depicted in Fig 4b) starts with a chaining activity $check\ weather$ resulting in output relation $warning$. This relation is the input relation for sending an SMS (activity $send\ SMS$), which is applied to every tuple individually (interval predicate $ID \in [maxID..maxID]$ and trigger predicate $LastSID < maxSID$). Both data workflows are well-formed.

The workflows described above (see Fig 4a and b) can be chained via activity $check\ weather$. The chaining operation is notated as $W_1 \uparrow W_2$ (see Def 6) and results in a workflow consisting of the union of relations and activities, where the annotations of activities contained in both workflows are taken from W_1.

Definition 6 (workflow chaining): Workflows W_1 and W_2 with $W_i = (\mathcal{R}_i, \mathcal{A}_i, \mathcal{F}_i, \tau_i, \iota_i, \kappa_i)$ and $i = \{1, 2\}$ can be chained $W = W_1 \uparrow W_2$ if

$\forall R \in \mathcal{R}_1 \cap \mathcal{R}_2 . \exists A \in \mathcal{A}_1 \cap \mathcal{A}_2 .$

$(A, R) \in \mathcal{F}_1 \bullet \cap \mathcal{F}_2 \bullet \wedge \kappa_1(A) \neq c \wedge \kappa_2(A) = c$

then $W = (\mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{F}_1 \cup \mathcal{F}_2, \tau, \iota_1 \cup \iota_2, \kappa)$

and $\kappa(A) = \begin{cases} \kappa_1(A) & if \quad A \in \mathcal{A}_1 \\ \kappa_2(A) & otherwise \end{cases}$

and $\tau(A) = \begin{cases} \tau_1(A) & if \quad A \in \mathcal{A}_1 \\ \tau_2(A) & otherwise \end{cases}$

□

The inverse to the chaining operation is the sub workflow operation notated as $W \downarrow \mathcal{R}'$ and defined in Def 7. This operation reduces workflow W to a workflow containing only relations in $\mathcal{R}'$ by chaining the reduced workflow to workflow W.

Definition 7 (sub workflow): From a workflow $W = (\mathcal{R}, \mathcal{A}, \mathcal{F}, \tau, \iota, \kappa)$ a sub workflow $W' \downarrow \mathcal{R}'$ over a set of relations $\mathcal{R}' \subseteq \mathcal{R}$ can be defined as
$W' \downarrow R' = (\mathcal{R}', \mathcal{A}', \mathcal{F}', \tau', \iota', \kappa')$ where
$\mathcal{A}' = \{A \in \mathcal{A} \mid R' \in \mathcal{R}' \wedge (R', A) \in \bullet\mathcal{F} \vee (A, R') \in \mathcal{F}\bullet\}$
$\mathcal{F}' = ((\mathcal{R}' \times \mathcal{A}') \cap \bullet\mathcal{F}) \cup ((\mathcal{A}' \times \mathcal{R}') \cap \mathcal{F}\bullet)$

$$\kappa'(A') = \begin{cases} det & if \quad \kappa(A) = det \wedge \exists R \in \mathcal{R}'.(R, A) \in \bullet\mathcal{F}' \\ c & if \quad \nexists R \in \mathcal{R}'.(R, A) \in \bullet\mathcal{F}' \\ \overline{det} & otherwise \end{cases}$$

$$\tau'(A') = \begin{cases} true & if \quad \kappa'(A) = c \\ \tau(A) & otherwise \end{cases}$$

$\iota' = \{F' \to \iota(F') \mid F' \in \mathcal{F}'\}$ □

Thus, for two workflows W_1 and W_2 and $\mathcal{R}'$ being the relations used in W_2 then the following equation holds: $W_2 = (W_1 \uparrow W_2) \downarrow \mathcal{R}'$.

V. Activity Interpretation

The syntax defined in the previous section is now given a semantics by formally defining an interpretation of data workflow concepts. Interpretation of trigger and interval predicates is done according to standard logic and arithmetic interpretations as e.g. in [25] where an interpretation is based on a valuation ν of variables. Variables are evaluated as natural numbers for a processing element using relational algebra expressions on a process state Σ.

The **process state** of a data workflow is the **information available in all relations** of the workflow (see Def 8. As a consequence a state change is inserting information (tuples) in a relation Rel at a specific point in time (transaction time). State changes are serialized for each relation. The state of the workflow is the union of state changes of all relations contained in the workflow. The information representing a single state change is called tuple element and is defined in Def 9.

Definition 8 (process state): A process state $\Sigma \subset TE$ is a finite set of tuple elements. □

Definition 9 (tuple element): A tuple element is a tuple (ID, TT, SID, Rel, t) where ID is an unique ID of the tuple t in a relation with name $Rel \in \mathcal{R}$, TT is the transaction time, i.e., the time when the tuple element has been created. SID is the tuple of all IDs of input relation tuple elements resulting in triggering activity A and producing the tuple element. The tuple t follows the relation schema of Rel or is an empty tuple ε. The set of all tuple elements is denoted by TE. □

The empty tuple, i.e. $t = \varepsilon$, indicates that a processing step has been executed without producing any output.

In the following the interpretation of all workflow concepts is introduced.

A. Activity

The interpretation of an activity A depends on the valuation of variables used in the annotations of the activity. Therefore, the valuations for variables introduced in the previous section are given as interpretations of relational algebra expressions applied to a process state Σ (see Def 10).

Definition 10 (variable valuation ν_A): Let A be an activity with $(R_i, A) \in \bullet F$ for $i = 1 \ldots n$ with interval predicate $I_i = \iota((R_i, A))$ and $(A, R) \in F\bullet$. Further let Σ be the current state. Then valuation $\nu_A(.) = \parallel . \parallel_{\nu_A}^{\Sigma,A}$ is

- variable maximum SID $maxSID$ is
  $\nu_A(maxSID) :=$
  $\prod_{i=1}^{n} max(\pi_{ID}(\sigma_{\parallel I_i \wedge TT \leq Now \wedge Rel = R_i \parallel_{\nu_A}^{\Sigma,A}}(\Sigma)))$.
- variable $LastTT$ represents transaction time of the last execution of activity A is
  $\nu_A(LastTT) := max(\pi_{TT}(\sigma_{\parallel TT < Now \wedge Rel = R \parallel_{\nu_A}^{\Sigma,A}}(\Sigma)))$.
- variable $LastSID$ represents the direct product of input relation IDs of the last execution of activity A is
  $\nu_A(LastSID) := max(\pi_{SID}(\sigma_{\parallel ID = LastID \parallel_{\nu_A}^{\Sigma,A}}(\Sigma)))$.
- variable $LastID$ represents the ID of the last execution of activity A is
  $\nu_A(LastID) := max(\pi_{ID}(\sigma_{\parallel TT < Now \wedge Rel = R \parallel_{\nu_A}^{\Sigma,A}}(\Sigma)))$.

The interpretation of Now is context dependent and therefore has to be specified explicitly. □

Given the variable valuation above, the interpretation of an activity can be defined as follows:

Definition 11 (interpretation activity): The interpretation $\parallel A \parallel_{\nu_A}^{\Sigma}$ of activity A with $(R_i, A) \in \bullet F$ for $i = 1 \ldots n$ and $\iota((R_i, A)) = I_i$ and $(A, R) \in F\bullet$ for state Σ with valuation ν_A specifies a set of tuple elements produced by activity A.
For input relations R_i the relevant state Σ_{R_i} for A is
$\Sigma_{R_i} = \sigma_{\parallel I_i \wedge TT \leq Now \wedge Rel = R_i \parallel_{\nu_A}^{\Sigma,A}}(\Sigma)$
For output relation R_o the relevant state Σ_{R_o} for A is
$\Sigma_{R_o} = \parallel A(\Sigma_{R_1}, \ldots, \Sigma_{R_n}) \parallel_{\nu_A}^{\Sigma}$
with $(id, tt, sid, R, t) \in \Sigma_{R_o}$ and

- $t \in \underline{A}(\Sigma_{R_1}, \ldots, \Sigma_{R_n})$
- $id = \nu_A(LastID) + pos(t)$ where $pos(t)$ is the position of tuple t in the result set produced by $\underline{A}()$
- $tt = \nu_A(Now)$
- $sid = \nu_A(maxSID)$

An interpretation of an activity is **complete** if
$\bigcup_{i=1}^{n} \Sigma_{R_i} \cup \Sigma_{R_o} = \sigma_{Rel \in \{R_1, \ldots, R_n, R_o\}}(\Sigma)$. □

Tab I provides some activities and their interpretations.

The notion of interpretation completeness of an activity (Def 11) can be extended to activity state completeness (Def 12). An activity is state complete, if for every interpretation of an activity represented in the process state the interpretation is complete.

Definition 12 (activity state completeness): Let A be an activity with input relations $(R_i, A) \in \bullet F$ for $i = 1 \ldots n$, interval predicates $\iota((R_i, A)) = I_i$, output relation $(A, R) \in F\bullet$, and trigger predicate $T = \tau(A)$. For all SIDs given as $sid \in \pi_{SID}(\sigma_{Rel=R}(\Sigma))$ a subset Σ_{sid} of state Σ is selected with $\Sigma_{sid} = \sigma_{SID \leq sid \wedge Rel = R}(\Sigma) \cup \bigcup_{i=1}^{n} \sigma_{ID \leq sid.i \wedge Rel = R_i}(\Sigma)$ and the valuation for Now is set as $\nu_A(Now) = \pi_{TT}(\sigma_{SID = sid \wedge Rel = R}(\Sigma))$.

A given process state Σ is **complete** for an activity A if for all SIDs (i) the trigger predicate T is interpreted as true, i.e. $\parallel T \parallel_{\nu_A}^{\Sigma_{sid},A} = true$, and (ii) the state Σ_{sid} provides a complete interpretation of activity A, i.e.
$\Sigma_{R_o} = \parallel A(\Sigma_{R_1}, \ldots, \Sigma_{R_n}) \parallel_{\nu_A}^{\Sigma_{sid}}$ is complete. □

If for every activity in a workflow activity state completeness holds, then the workflow is state complete (Def 13).

Definition 13 (workflow state completeness): A given process state Σ is complete for a workflow W if all activities in the workflow (i) are deterministic for activities with input relations or not deterministic for all other activities, and (ii) are state complete with Σ. □

B. Interval Predicate

The interpretation of an activity requires the interpretation of an interval predicate. A standard interpretation of a numerical expression is provided in Def 14.

Definition 14 (interpretation numerical expression): Let N be a numerical expression related to an activity A. The interpretation $\parallel N \parallel_{\nu_A}^{\Sigma,A}$ of N is a logical expression replacing variables V with their valuation ν_A.

- $\parallel N_1 \theta N_2 \parallel_{\nu_A}^{\Sigma,A} = \parallel N_1 \parallel_{\nu_A}^{\Sigma,A} \theta \parallel N_2 \parallel_{\nu_A}^{\Sigma,A}$ for numerical expressions with $\theta \in \{+, -, *\}$
- $\parallel N \parallel_{\nu_A}^{\Sigma,A} = N$ with $N \in \mathbb{R}$ being a number,
- $\parallel v \parallel_{\nu_A}^{\Sigma,A} = \nu_A(v)$ for a variable $v \in V$

□

The interpretation of the interval predicate (Def 15) results in a logical expression which is used in the interpretation of an activity for determining the state subset relevant from a particular input relation (see Def 11).

Definition 15 (interpretation interval predicate):
Let I be an interval predicate with $I = \iota((R, A))$ for an activity A. The interpretation $\parallel I \parallel_{\nu_A}^{A}$ of I is a logical expression replacing variable Now with its valuation ν_A and variable $maxID$ with $\nu_A(maxID) := max(\pi_{ID}(\sigma_{\parallel I'_i \wedge TT \leq Now \wedge Rel = R \parallel_{\nu_A}^{\Sigma,A}}(\Sigma)))$ where I'_i is derived from I_i by removing interval predicates containing $maxID$.

- $\parallel P_1 \wedge P_2 \parallel_{\nu_A}^{\Sigma,A} = \parallel P_1 \parallel_{\nu_A}^{\Sigma,A} \wedge \parallel P_2 \parallel_{\nu_A}^{\Sigma,A}$ for interval predicates P_1 and P_2

Activity name	Activity A	Interpretation $\underline{A}$	Constraint
Union	$Union(\Sigma_{R_1},\ldots,\Sigma_{R_n})$	$\bigcup_{i=1}^{n}\{t \mid (id,tt,sid,R_i,t)\in\Sigma_{R_i})\}$	schema $R_1,\ldots,R_n$ are equivalent
Selection	$Filter_P(\Sigma_{R_1})$	$\sigma_P(\{t \mid (id,tt,sid,R_1,t)\in\Sigma_{R_1}\})$	P is a valid predicate in SQL for R_1
Projection	$Map_m(\Sigma_{R_1})$	$\pi_m(\{t \mid (id,tt,sid,R_1,t)\in\Sigma_{R_1}\})$	m is a list of attribute names in R_1
No operation	$Noop(\Sigma_{R_1})$	$\{t \mid (id,tt,sid,R_1,t)\in\Sigma_{R_1}\}$	none
Average	$Avg(\Sigma_{R_1})$	$(\sum_{j=1}^{m} a_{j,1}/m,\ldots,\sum_{j=1}^{m} a_{j,k}/m)$ with $(a_{j,1},\ldots,a_{j,k})\in\{t \mid (id,tt,sid,R_1,t)\in\Sigma_{R_1}\}$	Average of all attributes in R

Table I
INTERPRETATION OF ACTIVITIES

- $\| TT \in (LB..UP] \|_{\nu_A}^{\Sigma,A} =$
  $TT > \| LB \|_{\nu_A}^{\Sigma,A} \wedge TT \leq \| UB \|_{\nu_A}^{\Sigma,A}$
- $\| ID \in (LB..UP] \|_{\nu_A}^{A} =$
  $ID > \| LB \|_{\nu_A}^{\Sigma,A} \wedge ID \leq \| UB \|_{\nu_A}^{\Sigma,A}$
- $\| true \|_{\nu_A}^{\Sigma,A} = true$
- $\| false \|_{\nu_A}^{\Sigma,A} = false$

□

C. Trigger Predicate

The interpretation of an activity requires the interpretation of a trigger predicate (Def 16), which is based on a standard interpretation of numerical expressions (Def 14). The interpretation produces an inequality and the notation $\| T \|_{\nu_A}^{\Sigma,A} = true$ is used to indicate that the inequality is valid.

Definition 16 (interpretation trigger predicate): Let T be a trigger predicate with $T = \tau(A)$ for an activity A. The interpretation $\| T \|_{\nu_A}^{\Sigma,A}$ of T for a state Σ and variables $V = \{Now, maxSID, LastTT, LastSID\}$ given in valuation ν_A results in an inequality derived by

- $\| LastTT\ \theta\ N \|_{\nu_A}^{\Sigma,A} = \| LastTT \|_{\nu_A}^{\Sigma,A}\ \theta\ \| N \|_{\nu_A}^{\Sigma,A}$ with $\theta \in \{<,\leq,=\}$ and numerical expression N
- $\| LastSID\ \theta\ N \|_{\nu_A}^{\Sigma,A} = \| LastSID \|_{\nu_A}^{\Sigma,A}\ \theta\ \| N \|_{\nu_A}^{\Sigma,A}$ with $\theta \in \{<,\leq,=\}$ and numerical expression N

□

VI. Data Workflow Interpretation

In the following the activity interpretation is extended to data workflows. Data workflow interpretation is comparable to a classical workflow execution semantics or a query processing semantics in databases. The algorithm describing the interpretation of a data workflow in this paper is the continuous data processing.

The coordination applied on controlling the interpretation of activities is based on the notion of activity state completeness (see Def 12), which each activity maintains locally by changing the process state. All workflow interpretations guarantee workflow state completeness (see Def 13) for snapshots in the processing. The interpretation is based on the assumption that data workflow processing is fast and therefore transaction time equals the time a measurement has been done. In future work this assumption will be relaxed.

Stream processing is characterized by data created by sensors or information systems, which are continuously propagated through a data workflow. The processing never terminates and the arrival of new data may trigger the interpretation of the activity "receiving" the data. The interpretation of activities is done in parallel, where each activity performs either Alg 1 or Alg 2 depending whether it is a time or tuple based trigger. For a time based trigger, at every point in time where the trigger predicate is interpreted as valid (line 1), the activity is interpreted and the corresponding state change is calculated (line 2). Then the current state is extended by the state change (line 3).

Algorithm 1: time triggered stream processing

```
Input: current state $\Sigma$
Output: state change $\Sigma_o = \emptyset$
1 foreach time $\nu(Now)$ with $\| T \|_{\nu_A}^{\Sigma,A} = true$ do
2     $\Sigma_o = \| A(\Sigma_{R_1},\ldots,\Sigma_{R_n}) \|_{\nu_A}^{\Sigma}$
3     $\Sigma = \Sigma \cup \Sigma_o$
```

For a tuple based trigger, it is much harder to predict when the next trigger will be enabled. Therefore, the algorithm (see Alg 2) is much more complicated than in for time based triggers. The algorithm continuously checks whether the set of new tuple elements TE determined in line 2 observed at a specific time (line 3) is sufficient to validate the trigger predicate as true (line 4). If this is the case then the activity is interpreted and the corresponding state change is calculated (line 5). Finally, the current state is extended by the state change (line 6).

Local state changes are possible, since each activity has exactly one output relation and each tuple element has the name of the relation included. Please be aware that this is a formal notation and an efficient implementation of the algorithm may make use of analyzing the trigger predicate.

VII. Optimization

Optimization means the re-organizing processing steps in a workflow optimizing a cost function while keeping the output of the workflow equivalent. As a consequence, before discussing optimization, the cost function and the notion of equivalence has to be clarified.

Algorithm 2: tuple triggered stream processing

```
Input: current state $\Sigma$
Output: state change $\Sigma_o = \emptyset$
1 Let $LastSID.i$ be the i-th element of tuple of IDs in
  variable $LastSID = \| LastSID \|_{\nu_A}^{\Sigma,A}$
2 foreach new tuple in any $R_i$, $1 \leq i \leq n$ do
3     $\nu_A(Now) = tt$ is the current time
4     if $\| T \|_{\nu_A}^{\Sigma \cup \Sigma_o, A} = true$ then
5         $\Sigma_o = \| A(\Sigma_{R_1}, \ldots, \Sigma_{R_n}) \|_{\nu_A}^{\Sigma \cup \Sigma_o}$
6     $\Sigma = \Sigma \cup \Sigma_o$
```

The idea of optimization as addressed in this paper aims at reducing computational power by sharing intermediate results between different workflows running on the same workflow engine. Therefore, the cost function is the utilization of a hardware. In particular, optimization aims at a homogeneous utilization of the hardware avoiding utilization peaks. A simple strategy could be: split the workload in as many as possible and as short as possible chunks, which should be processed as soon as possible [6]. The smaller the chunks the higher the likelihood that a same chunk requires processing by two workflows running on the same workflow engine. Sharing the processing result of this chunk reduces the utilization of the hardware.

An intuitive notion of equivalence of views is that two views are equivalent if their schemas are equivalent and if at any point in time the two views contain an equivalent set of tuples. Considering the fact that workflow optimization implies changing a workflow specification and therefore the time needed for the processing, it is almost impossible to guarantee that all tuple elements are available at the same time as it would be for the original workflow specification. Thus, applying such a strict notion of equivalence makes it impossible to perform any workflow optimization.

When investigating streaming data we can observe that an uncertainty principle applies: the more precise in time the processing of the data is performed, the less precise the data is due to processing delay. For example the processing of a daily average at 12:00 requires that all data collected until 11:59:59.999 are available at the processing step performing the average calculation. This is almost impossible and therefore the average result will be imprecise. If the average calculation is performed at 12:05 calculating the average for 12:00 all data will be available now, but the result is delayed by 5 minutes. This is the observed uncertainty principle. Applying this principle on an equivalence definition, means that the views must contain the same set of data, however, the time when the data gets available in the views may vary by a time difference δ.

[6]In this discussion the overhead of this splitting is not considered yet.

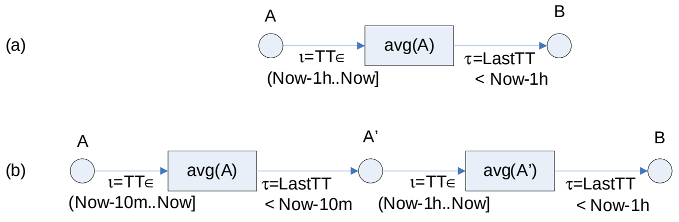

Figure 5. Transformation Rule for Workflow Optimization

Definition 17: Sets of tuples V and V' are called $delta$-equivalent at a point in time t iff $\forall r \in V.r.TT > t - \delta \vee \exists r' \in V'.r \equiv_\delta r'$ and with $\forall r' \in V'.r'.TT > t - \delta \vee \exists r \in V.r \equiv_\delta r'$ with $r \equiv_\delta r'$ iff each attribute of r and r' is equivalent except the attribute transaction time TT and $\mid r.TT - r'.TT \mid < \delta$. □

Applying this notions to views results in the following definition:

Definition 18: Two views are δ-equivalent if their schemas are equivalent and if at any point in time contain an δ-equivalent set of tuples. □

The δ used in the definition specifies the maximum allowed delay in processing time between two workflow specifications. Based on δ-equivalence several transformation rules can be defined. An example of such a transformation rule is depicted in Fig 5. Depending on the performance of the underlying hardware and the specified variance δ in processing times, the workflow specifications in Fig 5a) and b) are δ-equivalent. The basic idea behind the transformation is to base an one hour average calculation of data every hour produces the same output as a 10 minutes average calculated every 10 minutes, which is then further aggregated to hourly averages once an hour.

Applying this rule to the use case (see Fig 1 and 4a)) means that the calculated 15 minutes averages of the GPS coordinates for the workflow for Anna can be split into a 5 minutes average calculation, which is further aggregated to a 15 minutes average calculation afterwards. Since the workflow of Thomas and Anna then both contain the 5 minutes aggregate, the 5 minutes aggregation results can be shared between both workflows. This reduce the utilization of the hardware, and thus reduce the cost function.

Future work will investigate further transformation rules and algorithms to apply these rules.

VIII. Prototype

The formal definitions introduced in the previous sections have been implemented in a prototype. The prototype is based on a modular design as a basis for extending this prototype in future research.

Ffigure 6 illustrates the architecture. The diagram depicts three main layers: the process layer (top layer), the ap-

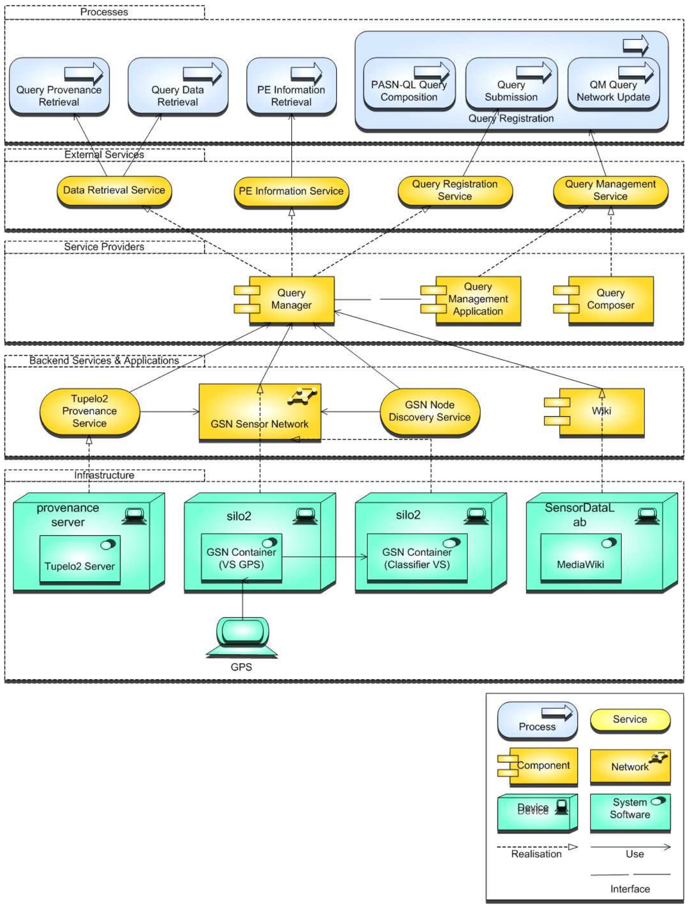

Figure 6. Prototype global system architecture

plication layer (middle layers), and the infrastructure layer (bottom layer).

A. *Processes*

The first group of objects consists of four main processes handled by the system. Provenance Retrieval allows in inquire on provenance information describing the origin of data contained in a view and the processing elements used to process these data. Query Data Retrieval allows to see the data of a particular view. Processing Element (PE) Information Retrieval allows to query on the current state of a processing element. Query registration consists of three subprocesses required for successfully registering a new query, i.e., a new processing element and its related output view: a new query needs to be composed and submitted, after which the query network is updated. Note that getting information about the network is probably also required for being able to compose a query, but since this process may also be used on its own, it has been excluded from the query registration process.

B. *Application Layer*

The application consists of three sub layers: external services, service providers, and backend services and applications.

External services There are four external services to enable the aforementioned processes. These services can be web services, but can also represent a software application or interface. In this case, the Data Retrieval Service, PE Information Service, and Query Registration Service are in fact web services, while the Query Management Service is a software or web application.

The **Service Providers**, i.e., the systems providing the external services, are modeled as components of the prototype. The dashed arrow stands for realization, and as such it can be seen that the Query Manager is responsible for most of the external services provided. The Query Management Application, which can be a software or web application, is an interface of the Query Manager for maintaining the query network and thus provides the query registration service. Finally, the Query Composer is a stand-alone tool that can aid the user in specifying queries in a visual way.

The **backend services & applications** are services and components that are not visible by the users of the system. First, there is a service for recording and querying provenance named Tupelo2 Provenance Service. The Tupelo2 provenance library [7] is used to implement this service, hence its name. Next, the GSN Sensor Network [8] is a component for acquiring streaming sensor data from various sensor types. This component clusters all GSN installations that can be facilitated as data sources by the Query Manager. The Query Manager will use the Node Discovery Service to find these containers. Finally, the SensorDataLab [9] Wiki is used as a data source of the Query Manager facilitating non streaming data integration. A wiki has been used as a placeholder for all kinds of enterprise information systems. It contains a lot of manually recorded and annotated data that can also be useful for the users of the system.

C. *Infrastructure*

The last layer depicts the infrastructure of the system. A separate provenance server is installed. Further, multiple GSN servers are running forming the GSN Sensor Network. The GPS sensor depicted indicates how sensors can be integrated on the infrastructure level facilitating GSN servers. The SensorDataLab Wiki is running on its own server.

One might notice that the Query Management Application and the Query Composer are not connected to any device in the infrastructure layer. The reason for this is that the device on which these components are running is unknown: it may be running on a client computer, but it can also be provided by a separate web application server.

[7]http://tupeloproject.ncsa.uiuc.edu/
[8]http://sourceforge.net/apps/trac/gsn/
[9]http://www.sensordatalab.org

IX. Conclusion

The presented data workflow model provides a workflow model for processing streaming data. The explication of data used in each processing step and the coordination mechanism in the data model enables a workflow engine to optimize the processing resources by sharing intermediate processing results between several workflow instances. Please be aware that the concepts and algorithms introduced require optimization when implemented. The proposed model has been implemented and the prototype is available as open source [10].

The next step is to investigate the effects of time constraints on transaction times and continue the illustrated work on data workflow optimization.

References

[1] B. Ludäscher, M. Weske, T. M. McPhillips, and S. Bowers, "Scientific workflows: Business as usual?" in *BPM*, ser. Lecture Notes in Computer Science, U. Dayal, J. Eder, J. Koehler, and H. A. Reijers, Eds., vol. 5701. Springer, 2009, pp. 31–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03848-8

[2] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Record*, vol. 34, no. 3, pp. 44–49, 2005. [Online]. Available: http://doi.acm.org/10.1145/1084805.1084814

[3] G. Kahn, "The semantics of simple language for parallel programming," in *IFIP Congress*, 1974, pp. 471–475.

[4] E. Lee and T. Parks, "Dataflow process networks," in *Proceedings of the IEEE*, vol. 83, 1995, pp. 773 – 801.

[5] B. Ludscher, J. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039 – 1065, 2005.

[6] "project web site," 2008, http://kepler-project.org/.

[7] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, T. Carver, M. Greenwood, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, June 2004. [Online]. Available: http://eprints.ecs.soton.ac.uk/10912/

[8] "project web site," 2008, http://taverna.sourceforge.net/.

[9] Y. Jararweh, A. Hary, Y. B. Al-Nashif, S. Hariri, A. Akoglu, and D. Jenerette, "Accelerated discovery through integration of kepler with data turbine for ecosystem research," *Computer Systems and Applications, ACS/IEEE International Conference on*, vol. 0, pp. 1005–1012, 2009.

[10] http://sourceforge.net/projects/sensordataweb/

[10] D. Barseghian, I. Altintas, M. B. Jones, D. Crawl, N. Potter, J. Gallagher, P. Cornillon, M. Schildhauer, E. T. Borer, E. W. Seabloom, and P. R. Hosseini, "Workflows and extensions to the kepler scientific workflow system to support environmental sensor data access and analysis," *Ecological Informatics*, vol. 5, no. 1, pp. 42–50, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.ecoinf.2009.08.008

[11] C. Wroe, C. A. Goble, A. Goderis, P. W. Lord, S. Miles, J. Papay, P. Alper, and L. Moreau, "Recycling workflows and services through discovery and reuse," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 2, pp. 181–194, 2007. [Online]. Available: http://dx.doi.org/10.1002/cpe.1050

[12] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," *Mobile Data Management, 2007 International Conference on*, pp. 198–205, May 2007.

[13] A. Salehi, "Global sensor network," 2008, http://gsn.sourceforge.net/.

[14] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, *Data-Stream Management: Processing High-Speed Data Streams*. Springer, 2006, ch. STREAM: The Stanford Data Stream Management System.

[15] "Stream," 2007, http://infolab.stanford.edu/stream/.

[16] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "Telegraphcq: Continuous dataflow processing for an uncertain world," in *CIDR*, 2003.

[17] "Telegraph," 2007, http://telegraph.cs.berkeley.edu/.

[18] J.-H. Hwang, Y. Xing, U. Cetintemel, and S. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," in *Proc. 23rd Intl Conf on Data Engineering (ICDE)*. IEEE Computer Society, 2007.

[19] "Borealis," 2007, http://www.cs.brown.edu/research/borealis.

[20] "Sql:2003," 2007, http://en.wikipedia.org/wiki/SQL:2003.

[21] M. Kifer, A. Bernstein, and P. M. Lewis, *Database Systems - An application-oriented approach*, 2nd ed. Pearson International Edition, 2006.

[22] W. Aalst, "Interorganizational workflows: An approach based on message sequence charts and petri nets," *Systems Analysis - Modelling - Simulation*, vol. 34, no. 3, pp. 335–367, 1999.

[23] A. Woodruff and M. Stonebreaker, "Supporting fine-grained data lineage in a database visualization environment," in *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*. Washington - Brussels - Tokyo: IEEE, Apr. 1997, pp. 91–103.

[24] WordNet, "a lexical database for the english language," http://wordnet.princeton.edu, 2004.

[25] J. Chomicki and G. Saake, Eds., *Logics for Database and Information Systems*. Kluwer, 1998.