DAGGTAX: A Taxonomy of Data Aggregation Processes

Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu

Abstract—Data aggregation processes are essential constituents for data management in modern computer systems, such as decision support systems and Internet of Things (IoT) systems. Due to the heterogeneity and real-time constraints in such systems, designing appropriate data aggregation processes often demands considerable efforts. A study on the characteristics of data aggregation processes will provide a comprehensive view for the designers, and facilitate potential tool support to ease the design process. In this paper, we propose a taxonomy called DAGGTAX, which is a feature diagram that models the common and variable characteristics of data aggregation processes, especially focusing on the real-time aspect. The taxonomy can serve as the foundation of a design tool that enables designers to build an aggregation process by selecting and composing desired features, and to reason about the feasibility of the design. We also provide a set of design heuristics that could help designers to decide the appropriate mechanisms for achieving the selected features. Our industrial case study demonstrates that DAGGTAX not only strengthens the understanding, but also facilitates the model-driven design of data aggregation processes.

Index Terms—data aggregation taxonomy, real-time data management, timeliness

1 Introduction

In modern information systems, data aggregation, defined as the process of producing a synthesized form from multiple data items [1], is commonly applied for data processing and management. For example, in order to discover unusual patterns and infer information, a data analysis application often computes a synthesized value from a subset of the database for statistical analysis [2]; in systems dealing with large amounts of data with limited storage, the data are often aggregated to save space [3]; in a sensor network, sensor data are aggregated, and only the aggregated data are transmitted so as to save bandwidth and energy [4]. Since data aggregation plays a key role in many applications, considerable research efforts have been dedicated to this topic. A number of taxonomies have been proposed to provide a comprehensive understanding on various aspects of data aggregation, such as aggregate functions ([1], [2], [5]), aggregation protocols ([4], [6], [7]) and security models ([8]).

The focus of this paper is instead on another important aspect: the data aggregation process (or DAP for short) itself. We consider a DAP as three ordered activities that allow raw data to be transformed into aggregated data via an aggregate function. First, a DAP starts with preparing the raw data needed for the aggregation from the data source into the aggregation unit called the aggregator. Next, an aggregate function is applied by the aggregator on the raw data, and produces the aggregated data. Finally, the aggregated data may be further handled by the aggregator, for example, to be saved into storage or provided to other processes. The main constituents of these activities are the raw data, the aggregate function and the aggregated data.

The main contribution of this paper is a global, high-level characterization of data aggregation processes. We justify our study of the DAP by the fact that it represents a pillar of an aggregation application's workflow, no matter if it is a centralized database management system or a highly distributed sensor network. Understanding DAP is essential to a correct design of the overall application. For instance, a sensor data gathering process, a data aggregation process and an analytic process form the basic workflow of a surveillance application. Multiple DAPs can also work together, one's aggregated data being another's raw data, to form a more complex, hierarchical aggregation process. To design a DAP, we must understand the desired features of its main constituents, that is, the raw data, the aggregate function and the aggregated data, as well as those of the DAP itself. Such features, ranging from functional features (such as data sharing) to extra-functional features (such as timeliness), are varying depending on different applications. One aspect of the understanding is to distinguish the mandatory features from the optional ones, so that the application designer is able to sort out the design priorities. Another aspect is to comprehend the implications of the features, and to reason about the (possible) impact on one another. Conflicts may arise among features, in that the existence of one feature may prohibit another one. Trade-offs should be taken into consideration at design time, so that infeasible designs can be ruled out at an early stage.

Among all features, we are particularly interested in the time-related properties of the DAP, since data aggregation is extensively applied in many real-time systems, such as automotive systems [9], avionic systems [10] and industrial automation [11]. In real-time systems, the correctness of a process depends on whether it completes on time, and validity of data depends on the time they are collected and accessed. These real-time properties are expected on raw data, aggregate function and the aggregated data, and impose constraints that cross-cut all three activities of a DAP. Therefore, we will especially emphasize the real-time related features and their implications.

In this paper we therefore propose a taxonomy of data aggregation processes, called DAGGTAX (Data AGGregation TAXonomy), with a focus on their features and consequent implications,

- *S. Cai, B. Gallina, D. Nyström and C. Seceleanu are with the Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden. Email: {simin.cai, barbara.gallina, dag.nystrom, cristina.seceleanu}@mdh.se*

from the perspective of the aggregation process itself. The proposed taxonomy is presented as a feature diagram [12]. The aim of our taxonomy is to ease the design of aggregation processes, by providing a comprehensive view on the features and cross-cutting constraints, with a systematic representation. The latter can serve as the basis of a design tool, which enables selecting the desired features, reasoning about possible trade-offs, reducing the design space of the application, and composing the features to build the desired aggregation processes.

The remaining part of the paper is organized as follows. In Section 2 we discuss the existing taxonomies of data aggregation. In Section 3 we present the preliminaries, followed by a survey of data aggregation processes in scientific literatures in Section 4. Section 5 presents the proposed taxonomy, and in Section 6 we introduce the design rules and heuristics based on the implications of the features presented in the taxonomy. In Section 7 we validate the taxonomy by a case study from industry. Section 8 gives a further discussion of the implications of the real-time features, before concluding the paper in Section 9.

2 Related Work

Many researchers have promoted the understanding of data aggregation on various aspects. Among these works, considerable efforts have been made on the study of aggregate functions. Mesiar et al. [13], Marichal [14], and Rudas et al. [1] have studied the mathematical properties of aggregate functions, such as continuity and stability, and discussed these properties of common aggregate functions in detail. A procedure for the construction of an appropriate aggregate function is also proposed by Rudas et al. [1]. In order to design a software system that computes aggregation efficiently, Gray et al. [2] have classified aggregate functions into distributive, algebraic and holistic, depending on the amount of intermediate states required for partial aggregates. Later, in order to study the influence of aggregate functions on the performance of sensor data aggregation, Madden et al. [5] have extended Gray's taxonomy, and classified aggregate functions according to their state requirements, tolerance of loss, duplicate sensitivity, and monotonicity. Fasolo et al. [4] classify aggregate functions with respect to four dimensions, which are lossy aggregation, duplicate sensitivity, resilience to losses/failures and correlation awareness. Our taxonomy builds on these works that focus on the aggregate functions mainly, and provide a comprehensive view of the entire aggregate processes instead.

A large proportion of existing works have their focus on in-network data aggregation, which is commonly used in sensor networks. In-network aggregation is the process of processing and aggregating data at intermediate nodes when data are transmitted from sensor nodes to sinks through the network [4]. Besides a classification of aggregate functions that we have discussed in the previous paragraph, Fasolo et al. [4] classify the existing routing protocols according to the aggregation method, resilience to link failures, overhead to setup/maintain aggregation structure, scalability, resilience to node mobility, energy saving method and timing strategy. The aggregation protocols are also classified by Solis et al. [7], Makhloufi et al. [6], and Rajagopalan [15], with respect to different classification criteria. In contrast to the above works focusing mainly on aggregation protocols, Alzaid et al. [8] have proposed a taxonomy of secure aggregation schemes that classifies them into different models. All these works differ from our taxonomy in that they provide taxonomies from a different perspective, such as network topology for instance. Instead, our work strives to understand the features and their implications of DAP and its constituents in design.

3 Preliminaries

In this section, we first recall the concepts of timeliness and temporal data consistency in real-time systems, after which we introduce feature models and feature diagrams that are used to present our taxonomy.

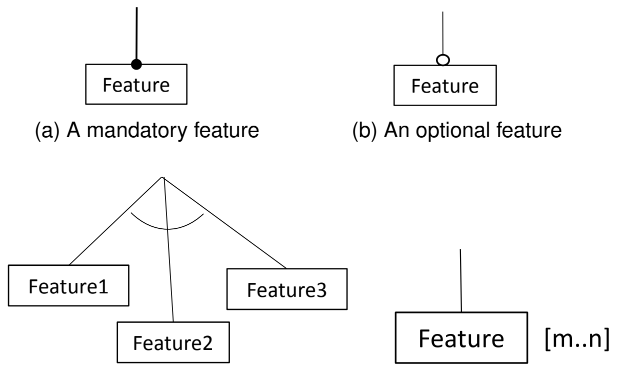
3.1 Timeliness and Temporal Data Consistency

In a real-time system, the correctness of a computation depends on both the logical correctness of the results, and the time at which the computation completes [16]. The property of completing the computation by a given deadline is referred to as *timeliness*. A real-time task can be classified as *hard*, *firm* or *soft* real-time, depending on the consequence of a deadline miss [16]. If a hard real-time task misses its deadline, the consequence will be catastrophic, e.g., loss of life or significant amounts of money. Therefore the timeliness of hard real-time tasks must always be guaranteed. For a firm real-time task, such as a task detecting vacant parking places, missing deadlines will render the results useless. For a soft real-time task, missing deadlines will reduce the value of the results. An example of soft real-time task is the signal processing task of a video meeting application, whose quality of service will degrade if the task misses its deadline.

Depending on the regularity of activation, real-time tasks can be classified as *periodic*, *sporadic* or *aperiodic* [16]. A periodic task is activated at a constant rate. The interval between two activations of a periodic task, called its *period*, remains unchanged. A sporadic task is activated with a *Minimum INter-arrival Time (MINT)*, that is, the minimum interval between two consecutive activations. During the design of a real-time system, a sporadic task is often modeled as a periodic task with a period equal to the MINT. A sporadic task may also have a *MAXimum inter-arrival Time (MAXT)* which specifies the maximum interval between two consecutive activations. An aperiodic task is activated with an unpredictable interval between two consecutive activations. A task triggered by an external event with unknown occurrence pattern can be seen as aperiodic.

Real-time applications often monitor the state of the environment and react to changes accordingly and timely. The environment state is represented as data in the system, which must be updated according to the actual environment state. The coherency between the value of the data in the system and its corresponding environment state is referred to as *temporal data consistency*, which includes two aspects, the *absolute temporal validity* and *relative temporal validity* [17]. A data instance is absolute valid, if the timespan between the time of sampling its corresponding real-world value, and the current time, is less than a specified *absolute validity interval*. A data instance derived from a set of data instances (base data) is absolute valid if all participating base data are absolute valid. A derived data instance is relative valid, if the base data are sampled within a specified interval, called *relative validity interval*.

Data instances that are not temporally consistent may lead to different consequences. Different levels of strictness with respect to temporal consistency thus exist, which are *hard*, *firm* and *soft* real-time, in a decreasing order of strictness. Using outdated hard real-time data could cause disastrous consequences, and therefore



(a) A mandatory feature (b) An optional feature

(c) A group of alternative features (d) A feature with cardinality

Fig. 1. Notations of a feature diagram

this should not appear. Firm real-time data are useless if they are outdated, whereas outdated soft real-time data can still be used, but will yield degraded usefulness.

3.2 Feature Model and Feature Diagram

The notion of *feature* was first introduced by Kang et al. in the Feature-Oriented Domain Analysis (FODA) method [12], in order to capture both the common characteristics of a family of systems as well as the differences between individual systems. Kang et al. define a feature as a prominent or distinctive system characteristic visible to end-users. Czarnecki and Eisenecker extend the definition of a feature to be any functional or extra-functional characteristic at the requirement, architecture, component, or any other level [18]. This definition allows us to model the characteristics of data aggregation processes as features. A *feature model* is a hierarchically organized set of features, representing all possible characteristics of a family of software products. A particular product can be formed by a combination of features, often called a configuration, selected from the feature model of its family.

A feature model is usually represented as a *feature diagram* [12], which is often depicted as a multilevel tree, whose nodes represent features and edges represent decomposition of features. In a feature diagram, a node with a solid dot represents a common feature (as shown in Fig. 1a), which is mandatory in every configuration. A node with a circle represents an optional feature (Fig. 1b), which may be selected by a particular configuration. Several nodes associated with a spanning curve represent a group of alternative features (Fig. 1c), from which one feature must be selected by a particular configuration. The cardinality [m..n] ($n \geq m \geq 0$) annotated with a node in Fig. 1d denotes how many instances of the feature, including the entire sub-tree, can be considered as children of the feature's parent in a concrete configuration. If $m \geq 1$, a configuration must include at least one instance of the feature, e.g., a feature with [1..1] is then a mandatory feature. If $m=0$, the feature is optional for a configuration.

A valid configuration is a combination of features that meets all specified constraints, which can be dependencies among features within the same model, or dependencies among different models. An example of such a constraint is that the selection of one feature requires the selection of another feature. Researchers in the software product line community have developed a number of tools, providing extensive support for feature modeling and the verification of constraints. For instance, in FeatureIDE [19], software designers can create feature diagrams using a rich graphic interface. Designers can specify constraints across features as well as models, to ensure that only valid configurations are generated from the feature diagram.

4 A Survey of Data Aggregation Processes

Serving as an important information processing and analysis technique, data aggregation has been widely applied in a variety of information management systems. Based on scientific literature, in this section, we present a limited survey of application examples that implement data aggregation processes. In order to extract heuristics that help us generate our taxonomy, we select the examples from a wide variety of application domains, and investigate the common and different characteristics of aggregation processes. Some of these examples are general-purpose infrastructures that implement aggregation as a basic service. The other examples develop data aggregation as ad hoc solutions suitable for the particular application scenarios.

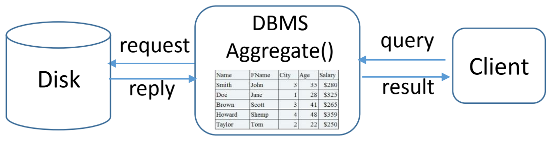
In the following subsections, we first present how aggregation is supported in different general-purpose infrastructures that provide data processing and management. Next, a number of ad hoc applications are presented, focusing on the requirements that the aggregation processes implemented in such applications must meet. Finally, we discuss the characteristics of aggregation processes exposed in the surveyed systems and applications.

4.1 General-purpose Infrastructures

In this subsection, we investigate the design of aggregation processes in general-purpose systems from the following domains: database management systems, data warehouses, data stream management systems and wireless sensor networks.

Database Management Systems and Data Warehouses: Many information management systems adopt a general-purpose relational Database Management System (DBMS) or a Data Warehouse (DW) [20] as a back-end for centralized data management, which have common aggregate functions implemented, and exposed as interfaces for users or programmers. Internally, aggregation is supported by a number of infrastructural services, including query evaluation, data storage and accessing, trigger mechanism, and transaction management. In a typical disk-based relational DBMS, as illustrated in Fig. 2, data are stored as tuples in the disk. An aggregation process is started by a query issued by a client. The DBMS then evaluates the query and loads the relevant tuples from disk into the main memory. An aggregate function is performed on the tuples and computes the aggregated value, which is then returned to the query issuer, cached in main memory or stored in the disk. An aggregation process can also be triggered by a state change in the database. Both raw data and aggregated data can be accessed by other processes. In order to maintain logical data consistency, such processes, including the aggregation process, are treated as transactions and governed by the transaction management system, which ensures the so-called ACID (Atomicity, Consistency, Isolation, and Durability) properties [21] during their executions.

Data can be aggregated by categories, usually specified in the "group-by" clause of a query. These categories may have a hierarchical relationship and thus represent the granularities of aggregation. For example, in a temporal database, users may choose to aggregate data by day, week or month, with a coarser granularity; in a spatial database, the aggregation can be based on streets, cities and provinces [22]. In a data warehouse, the stored



Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Fig. 2. Illustration of aggregation in a disk-based DBMS

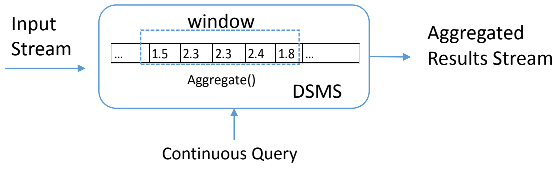


Fig. 3. Illustration of aggregation in a data stream management system

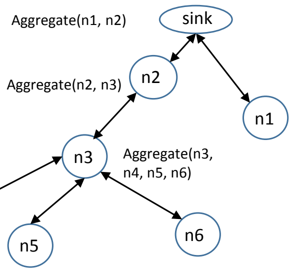


Fig. 4. Illustration of aggregation in a wireless sensor network

data usually have many dimensions, and the aggregation may need to be performed on multiple dimensions [20].

The aggregated value may be returned to the query issuer directly, or may be stored persistently in the database as a normal tuple. Alternatively, the aggregated values are cached in materialized views, so that other processes can make use of them [23]. It is common to store the aggregated values as materialized views in data warehouse since these results will be frequently used by analysis processes [20].

A number of aggregate functions are included in the SQL standard and are commonly supported by general-purpose DBMSs. Other aggregate functions can be defined as user-defined functions. The aggregation can be triggered by an explicit query issued by the client, or by a trigger that reacts to the change of the database. In a data warehouse, aggregation is often planned periodically to refresh the materialized views using the updated base data. In case a query needs to access current data between the planned aggregation processes, extra aggregation processes may also be started to refresh the views [20].

Online Aggregation in Data Stream Management Systems: Data aggregation in traditional DBMSs and DWs is performed like batch-processing: on a large number of tuples and in considerable time before returning the aggregated value. To improve performance and user experience, Hellerstein et al. propose “online aggregation” [24], which allows tuples to be aggregated incrementally. Tuples are selected from a base table by a sampling process, and aggregated with the cached partial aggregated result from previously sampled tuples. The partially aggregated value is available, which refers to the user as an approximate aggregated result. The aggregation process is defined with a stopping interface, through which the aggregation can be stopped, giving the approximate result as the final result.

Online aggregation is often supported by Data Stream Management Systems (DSMSs), which provide centralized aggregation for continuous data streams. In Fig. 3, we illustrate the aggregation in a typical DSMS scenario. Usually, stream data are pushed into the DSMS continuously, often at a high frequency. Individual data instances are not significant, become stale as time passes, and do not need to be stored persistently. Finite subsets of the most recent incoming stream (“windows”) are cached in the system. Aggregate functions can be defined by users and are applied on the windows. In the Aurora data stream management system [25], the aggregate function can be associated with a “timeout” parameter, indicating the deadline of the computation of the function. A function should return before it times out, even if some raw data instances are missing or delayed, so as to provide timely response required by many real-time applications. Aurora has implemented a load shedding mechanism, which drops data instances when the system is overloaded. The aggregation is triggered either by continuous queries with specified periods, or by ad hoc queries which are issued by clients. The aggregated results are passed to the receiving application as an outgoing stream. To provide historical data, the aggregated data may also be kept persistently for a specified period of time.

Multiple aggregation processes can be run concurrently, performing aggregation on the same data stream [26]. Oyamada et al. [27] point out that the aggregation in a DSMS may also involve non-streaming data, which can be shared and updated by other processes, causing potential data inconsistency. The authors propose a concurrency control mechanism to prevent the inconsistency.

In-network Aggregation in Wireless Sensor Networks: Data aggregation plays an essential role in Wireless Sensor Network (WSN) applications. In these applications, numerous data are gathered from resource-constrained sensor nodes that are deployed to monitor the environment. The gathered data are transmitted through a network to sink nodes, which are equipped with more resource for advanced computation and analysis. Along the transmission, data are aggregated in the intermediate sensor nodes or special aggregate nodes, in a decentralized topology. This aggregation technique is also called “in-network aggregation” [4]. In contrast, a sensor network can also apply centralized aggregation if the data of all sensors are transmitted to and aggregated in one single node. Fig. 4 gives an example of data aggregation in a sensor network. In this example, data from nodes n4, n5 and n6 are aggregated in node n3. This aggregated result is then transmitted to n2, and aggregated with the data of n2. Finally, the data from n2 and n1 are aggregated in the sink node.

Madden et.al [5] propose Tiny AGgregation (TAG), a generic aggregation service for ad hoc networks. In TAG, the user poses aggregation queries from a base station, which are distributed to the nodes in the network. Sensors collect data and route data back to the base station through a routing tree. As the data flow up the tree, it is aggregated by an aggregation function and value-based partitioning according to the query, level by level. At each level, a node awakens when it receives the aggregate request, together with a deadline when it should reply to its parent, and propagates the request to its children with an earlier deadline. Each node then listens to its children, aggregates the data transmitted from the children and the reading of itself, and then replies the aggregated result to its parent. If any node replies after its specified deadline, its value will not be aggregated by its parent, which means that the final aggregated result is actually an approximation.

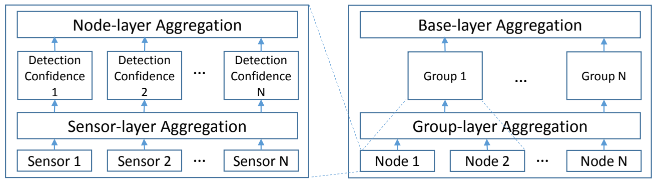


Fig. 5. Data Aggregation Architecture of VigilNet [29]

The aggregated results are cached by the nodes, and can be used for fault tolerance reason, e.g., loss of messages from a child. TAG has also classified aggregate functions into distributive, algebraic, holistic, unique and content-sensitive. Decentralized in-network aggregation is only appropriate for distributive and algebraic aggregate functions, since they can be decomposed into sub-aggregates. For other functions, all sensor data have to be collected to one node and aggregated together.

TAG is later implemented in the TinyDB [28], which supports SQL-style queries. Aggregation can be triggered periodically by continuous queries, or at once by a state change or an ad hoc query. Aggregated results can be stored persistently as storage points, which may be accessed by other processes.

4.2 Ad Hoc Applications

Many applications have unique requirements, and consequently use their ad hoc aggregation processes to fulfill their requirements. Examples of such applications are presented in the following paragraphs.

He et al. present the VigilNet for real-time surveillance with a tiered architecture [29]. Four layers are implemented in this system and each layer has its data aggregation requirements. The data aggregation architecture of VigilNet is illustrated in Fig. 5. The first layer is the sensor layer in which data inputs are pushed from individual sensors at specific rates, and aggregated as detection confidence vectors. In this layer the aggregation needs to meet stringent real-time constraints since the sensors send signals about fast-moving targets. The results of sensor-layer aggregation are sent to the node for node-layer aggregation. Each sensor node includes several sensors, and computes the average of sensor confidence vectors incrementally when a new sensor confidence vector arrives. If the aggregated results show the existence of a tracking target, the node estimates the position of the target, and sends a report to the leading node of the local group. The leader buffers the reports from members, until the number reaches a predefined aggregation degree. Then, it aggregates all the reports, estimates the current position of the target, and sends the aggregated report to the base station. The base station aggregates the new report with historical positions of the target, and calculates the velocity using a linear regression procedure.

Defude et al. propose the VESPA (Vehicular Event Sharing with a mobile P2P Architecture) approach [30] for the Vehicular Ad hoc NETwork (VANET), to aggregate traffic information events, such as parking places, accidents and road obstacles, pushed from neighbor vehicles. The events are aggregated by times, areas and event types. The aggregated values are stored and accessed for further analysis.

Goud et al. [9] propose a real-time data repository for automotive adaptive cruise control systems. It includes an Environment Data Repository (EDR) and a Derived Data Repository (DDR). The EDR periodically reads sensor readings, aggregates them, and keeps the aggregated value in the repository. The DDR then reads and aggregates the values from EDR, only when the changes of readings from some sensors exceed a threshold. The sensor data are real-time and have their validity intervals. The aggregate processes must complete before the data become invalid, and produce the results for other processes with stringent deadlines.

Arai et al. propose an adaptive two phase approach for approximate ad hoc aggregation in unstructured peer-to-peer (P2P) systems [31]. When an ad hoc aggregate query is issued, in the first phase, sample peers are visited by a random walk from the sink, with a predefined depth. Information of the visited peers are collected to the sink, and analyzed to decide the peers to be aggregated. These peers are then visited in the second phase. For some aggregate functions such as COUNT and AVERAGE, partial aggregate results are computed in the local peer, and returned to the sink. For other aggregate functions, raw data are returned to the sink and aggregated in the sink.

Baulier et al. [32] propose a database system for real-time event aggregation in telecommunication systems. Events generated by phone calls are pushed into the system, which should be aggregated within specific response times. The aggregated results are kept in a main-memory database as views for other time-critical processes. When a new event arrives, it triggers the aggregate process to update the aggregate view. The event record itself is stored into a data warehouse persistently, which is not time-critical.

Bar et al. [33] propose an online aggregation system for network traffic monitoring where large volumes of heterogeneous data streams are processed with different time constraints. Arriving stream data instances, as well as non-stream data, are stored persistently in the system. Aggregation can be triggered by ad hoc queries, or triggered periodically by continuous queries. The aggregate results are stored persistently in materialized views. Aggregate functions are computed incrementally, by combining the newly arrived instance with cached aggregated results.

Bür et al. describe an online active control system for aircrafts which employs data aggregation [10]. In this application, real-time data are gathered periodically from sensors deployed in the aircraft, and aggregated periodically. Since the aircraft system is time-critical, the freshness of data and timely processing of aggregation are crucial.

Lee et al. propose an approach for aggregating data in an industrial manufacturing system [11]. Three types of aggregation are described, which are aggregation at device level, aggregation in control system, and aggregation in remote monitoring system. At device level, real-time raw data are produced by sensors and controllers, and are aggregated in the devices. The aggregation is triggered hourly, or by state changes in the device. The aggregation functions are simple calculations for hourly throughput, error count, etc. The aggregated values are sent to subscribing clients, namely the control system and the remote monitoring system. The control system receives the data from devices and store them into a database. Every hour, these data, together with other events, are aggregated to produce error times, throughput, etc. The remote monitoring system also stores the data from devices and performs aggregation. Delay could occur in aggregation in the remote system.

Iftikhar applies data aggregation on integration of data in farming systems [3]. Data are collected from different devices, and stored permanently in a relational database. A gradual granular data aggregation strategy is then applied on the stored data.

Basically, older data should be aggregated in a coarse-grained granularity while newer data are aggregated in a finer granularity. For different granularities, aggregation is triggered in different periods. The aggregated results are kept in the database while the raw data are deleted to save space.

Golab et al. propose a tool called DataDepot for generating data warehouses from streaming data feeds [34], focusing on the real-time quality of the data. Raw data are modeled as tables, which are not persistent and have a freshness property. Raw data are generated from different sources, with various properties such as rate and freshness. Raw tables are aggregated and stored in persistent derived tables which must also be fresh. Updates in the raw tables are propagated to the derived tables.

4.3 Survey Results

More than 13,000 research works are indexed in the SCOPUS search engine using "data aggregation" as a search key for title, abstract and keywords in computer science and engineering. Although only a small proportion of related works are examined here, our survey covers a relevant set of systems and application domains, which exposes the common and variable characteristics of the raw data, aggregated data, the aggregate functions, as well as the entire data aggregation processes.

In Table 1, we summarize the previous review by listing characteristics of the DAPs in the surveyed systems and applications. Clearly, each aggregation process must have raw data, an aggregation function and the aggregated results. However, other characteristics have shown great variety. For instance, aggregation processes prepare the raw data ready for aggregation, by different data acquisition schemes. In some applications the aggregation process needs to pull the raw data from the persistent storage of the data source. Therefore the designer of an aggregation process must take this interaction into consideration. In other applications, however, raw data are pushed by the data source, so fetching raw data is not the concern of the aggregation process. The aggregated data may be stored persistently in some scenarios and are expected to survive system failures, while in other scenarios they can only reside in the volatile memory. As one can see in Table 1, the consistency of the data may depend on the time in some DAPs, while in others the data are static. A large variety of aggregate functions have been applied in aggregation processes, depending on the requirements of the particular application. The aggregation process itself may be scheduled periodically, or triggered by ad hoc events. In time-critical systems, the aggregation processes have strict timeliness requirements, while in some analytical systems with large amount of data the delays of the aggregation processes are tolerable. To design an appropriate aggregation process, it follows that one must take these characteristics, as well as their nature (necessity, optionality, etc) and their cross-cutting constraints, into consideration. A designer could benefit from having a systematic representation of these characteristics to ease the design, as well as support for facilitating feasible choices of the involved characteristics.

5 Our Proposed Taxonomy

The survey presented in Section 4 has revealed a number of characteristics of aggregation processes, including the raw data, the aggregated data, the aggregate functions, as well as the triggering patterns and the timeliness of the processes. Some of these characteristics are common for all aggregation processes, while others are distinct from case to case. In this section we propose a taxonomy of data aggregation processes, as an ordered arrangement of features revealed by the survey. The taxonomy for these common and variable characteristics not only leads to a clear understanding of the aggregation process, but also lays a solid foundation for an eventual tool support for reasoning about the impact of different features on the design.

We choose feature diagram as the presentation of our taxonomy, mainly due to two reasons. First, features may be used to model both functional and extra-functional characteristics of systems. This allows us to capture cross-cutting aspects that have on multiple software modules related to different concerns. Second, the notation of feature diagrams is simple to construct, powerful to capture the common and variable characteristics of different data aggregation processes, and intuitive to provide an organizational view of the processes. The taxonomy is shown in Fig. 6.

In the following subsections, these features are discussed in details with concrete examples. More precisely, the discussion is organized in order to reflect the logical separation of features. We explain Fig. 6 from the top level features under "Aggregation Process", and iterate through all sub-features in a depth-first way. The top level features include "Raw Data", "Aggregate Function" and "Aggregated Data", which are the main constituents of an aggregation process. Features that characterize the entire DAP are also top level features, including the "Triggering Pattern" of the process, and "Real-Time (P)", which refers to the timeliness of the entire process.

5.1 Raw Data

One of the mandatory features of real-time data aggregation is the raw data involved in the process. Raw data are the data provided by the DAP data sources. One DAP may involve one or more types of raw data. The multiplicity is reflected by the cardinality [1..*] next to the feature "**Raw Data Type**" in Fig. 6. Each raw data type may have a set of raw data. For instance, a surveillance system has two types of raw data ("sensor data" and "camera data"), while for the sensor data type there are several individual sensors with the same characteristics. Each raw data may have a set of properties, which are interpreted as its sub-features and constitute a sub-tree. These sub-features are: Pull, Shared, Sheddable, and Real-Time.

Pull: "Pull" is a data acquisition scheme for collecting raw data. Using this scheme, the aggregator actively acquires data from the data source, as illustrated in Fig. 7a. For instance, a traditional DBMS adopts the pull scheme, in which raw data are acquired from disks using SQL queries and aggregated in the main memory.

"Pull" is considered to be an optional feature of raw data, since not every DAP pulls data actively from data source. If raw data have the "pull" feature, pulling raw data actively from the data source is a necessary part of the aggregation process, including the selection of data as well as the shipment of data from the data source. If the raw data do not have the "pull" feature, they are pushed into the aggregator (Fig. 7b). In this case, in our view the action of pushing data is the responsibility of another process outside of the DAP. From DAP's perspective, the raw data are already prepared for aggregation.

"**Persistently Stored**" is considered as an optional sub-feature of "Pull", since raw data to be pulled from data source may be stored persistently in a non-volatile storage, such as a disk-based

TABLE 1
Characteristics of Data Aggregation Processes in the Surveyed Applications

Sample	Raw Data	Aggregate Function	Aggregated Data	Triggering Pattern	Real-time Characteristics
relational disk-based DBMS/DW [20], [22], [23]	pulled from data sources; persistently stored; possibly shared by other processes	various functions	possibly durable; possibly shared by other processes	activated by events (queries or database triggers), or activated periodically	usually no deadlines
DSMS (AURORA [25], Oyamada et al. [27], Krishnamurthy et al. [26])	pushed by data sources; possibly pushed periodically; not persistently stored; cached for a particular period; real-time; possibly shared by other processes; can be shedded	various functions	pushed to receiver; possibly durable; may be stored for a period of time	activated by events (ad hoc queries), or activated periodically (periodic continuous queries)	deadlines depending on the application
WSN (TAG [5], TinyDB [28])	pulled from data sources; not persistently stored; possibly be skipped	various functions	cached for a particular period; possibly durable; real-time; possibly shared by other processes	activated by events, or activated periodically	deadlines depending on the application
VigilNet [29], sensor layer	pushed by data sources; not persistently stored; real-time; pushed periodically	detection confidence function	pushed to receiver; not durable	activated periodically	hard deadlines
VigilNet [29], node layer	pushed by data sources; not persistently stored	average	pushed to receiver; not durable	activated by event	soft deadlines
VigilNet [29], group layer	pushed by data sources; cached for a particular period	ad hoc calculation	pushed to receiver; not durable	activated by event	soft deadlines
VigilNet [29], base layer	pushed by data sources; persistently stored	regression	shared by other processes	activated by event	soft deadlines
VESPA [30]	pushed by data sources	various functions	durable; shared by other processes	activated by events	soft deadlines
Goud et al. [9], EDR	pulled from data sources; pulled periodically; real-time; not persistently stored	various functions	not durable; real-time; shared by other processes	activated periodically	hard deadlines
Goud et al. [9], DDR	pulled from data sources; real-time; not persistently stored	various functions	durable; real-time	activated by events	hard deadlines
Arai et al. [31]	pulled from data sources; not persistently stored	various functions	possibly durable	activated by events	no deadlines
Baulier et al. [32]	pushed by data sources; persistently stored	various functions	real-time; not durable; shared by other processes	activated by events	hard deadlines
Bar et al. [33]	pushed by data sources; persistently stored; possibly real-time	various functions	durable	activated by events, or activated periodically	soft deadlines
Bür et al. [10]	pushed by data sources; not persistently stored; real-time;	various functions	not durable; real-time	activated periodically	hard deadlines
Lee et al. [11], device level	pushed by data sources; real-time	various functions	pushed to receiver; not durable	activated by events, or activated periodically	soft deadlines
Lee et al. [11], control system	pulled from data sources; persistently stored	various functions	possibly durable	activated periodically	soft deadlines
Lee et al. [11], remote monitoring system	pulled from data sources; persistently stored	various functions	possibly durable	activated periodically	soft deadlines
Iftikhar [3]	pulled from data sources; persistently stored; stored for a particular period; possibly shared by other processes	various functions	durable; stored for a particular period; possibly shared by other processes	activated periodically	soft deadlines
DataDepot [34]	pulled from data sources; not persistently stored; possibly shared by other processes; real-time	various functions	durable; real-time	activated by events	deadlines depending on the application

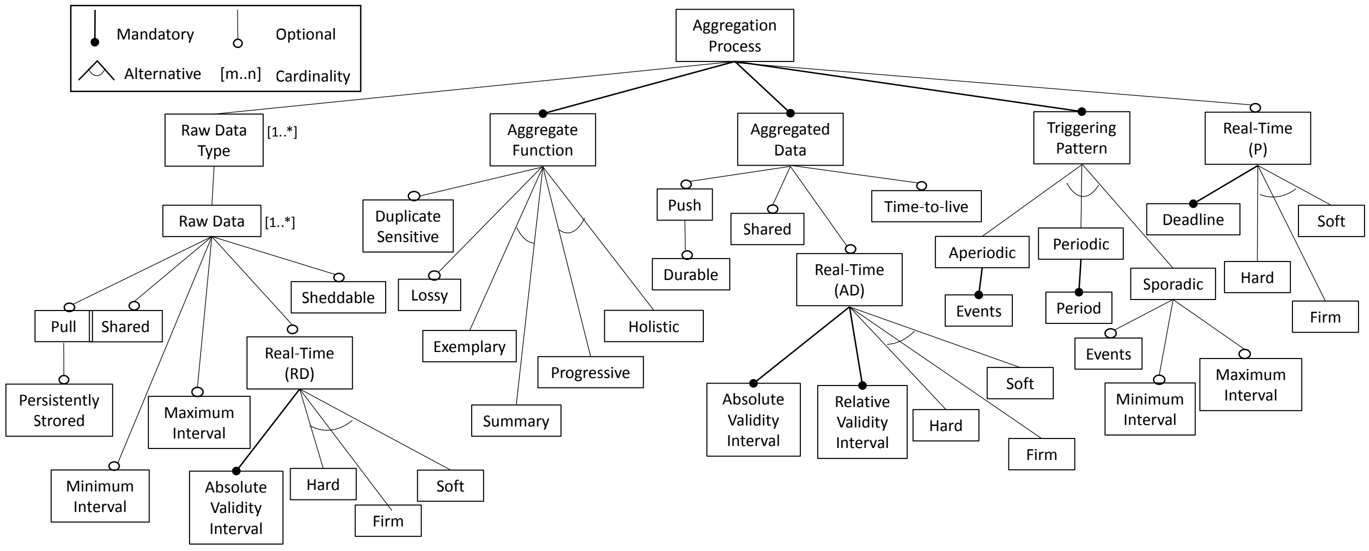


Fig. 6. The taxonomy of data aggregation processes

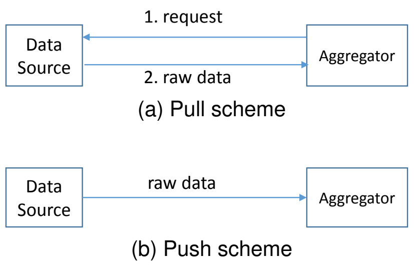


Fig. 7. Raw data acqusition schemes

relational DBMS. The retrieval of persistent raw data involves locating the data in the storage and the necessary I/O.

Shared: Raw data of some DAP examples in Section 4 are read or updated by other processes at the same time when they are read for aggregation [3], [26], [27]. The same raw data may be aggregated by several DAPs, or accessed by processes that do not perform aggregations. We use the optional "shared" feature to represent the characteristic that the raw data involved in the aggregation may be shared by other processes in the system.

Sheddable: We classify the raw data as "sheddable", which is an optional feature, used in cases when data can be skipped for the aggregation. For instance, in TAG [5], the inputs from sensors will be ignored by the aggregation process if the data arrive too late. In a stream processing system, new arrivals may be discarded when the system is overloaded [25]. For raw data without the sheddable feature, every instance of the raw data is crucial and has to be computed for aggregation.

Real-Time (RD): The raw data involved in some of the surveyed DAPs have real-time constraints. Each data instance is associated with an arrival time, and is only valid if the elapsed time from its arrival time is less than its **absolute validity interval**. "Real-time" is therefore considered an optional feature of raw data, and "absolute validity interval" is a mandatory sub-feature of the "real-time" feature. We name the real-time feature of raw data as "Real-Time (RD)" in our taxonomy, for differentiating from the real-time features of the aggregated data ("Real-Time (AD)" in Section 5.3) and the process ("Real-Time (P)" in Section 5.5).

Raw data with real-time constraints are classified as "**hard**", "**firm**" or "**soft**" real-time, depending on the strictness with respect to temporal consistency. They are represented as alternative sub-features of the real-time feature. As we have explained in Section 3, hard real-time data (such as sensor data from a field device [11]) and firm real-time data (such as surveillance data [29]) must be guaranteed up-to-date, while outdated soft real-time data are still of some value and thus can be used (e.g., the derived data from a neighboring node in VigilNet [29]).

MINT: Raw data may arrive continuously with a Minimum INter-arrival Time (MINT), of which a fixed arrival time is a special case. For instance, in the surveillance system VigilNet [29], a magnetometer sensor monitors the environment and pushes the latest data to the aggregator at a frequency of 32HZ, implying a MINT of 32.15 milliseconds. We consider "MINT" an optional feature of the raw data.

5.2 Aggregate Function

An aggregation process must have an aggregate function to compute the aggregated result from raw data. An aggregate function exhibits a set of characteristics that we interpret as features.

Duplicate Sensitive: "Duplicate sensitivity" has been introduced as a dimension by Madden et al. [5] and Fasolo et al. [4]. An aggregate function is duplicate sensitive, if an incorrect aggregated result is produced due to a duplicated raw data. For example, COUNT, which counts the number of raw data instances, is duplicate sensitive, since a duplicated instance will lead to a result one bigger than it should be. MIN, which returns the minimum value of a set of instances, is not duplicate sensitive because its result is not affected by a duplicated instance. "Duplicate sensitive" is considered as an optional feature of the aggregate function.

Exemplary or Summary: According to Madden et.al [5], an aggregate function is either "exemplary" or "summary", which are represented as alternative features in our taxonomy. An exemplary aggregate function returns one or several representative values of the selected raw data, for instance, MIN, which returns the minimum as a representative value of a set of values. A summary aggregate function computes a result based on all selected raw data, for instance, COUNT, which computes the cardinality of a set of values .

Lossy: An aggregate function is "lossy", if the raw data cannot be reconstructed from the aggregated data alone [4]. For example, SUM, which computes the summation of a set of raw data instances, is a lossy function, as one cannot reproduce the raw data instances from the aggregated summation value without any additional information. On the contrary, a function that concatenates raw data instances with a known delimiter is not lossy, since the raw data can be reconstructed by splitting the concatenation. Therefore, we introduce "lossy" as an optional feature of aggregate functions.

Holistic or Progressive: Depending on whether the computation of aggregation can be decomposed into sub-aggregations, an aggregate function can be classified as either "progressive" or "holistic". The computation of a progressive aggregate function can be decomposed into the computation of sub-aggregates. In order to compute the AVERAGE of ten data instances, for example, one can compute the AVERAGE values of the first five instances and the second five instances respectively, and then compute the AVERAGE of the whole set using these two values. The computation of a holistic aggregate function cannot be decomposed into sub-aggregations. An example of holistic aggregate function is MEDIAN, which finds the middle value from a sequence of sorted values. The correct MEDIAN value cannot be composed by, for example, the MEDIAN of the first half of the sequence together with the MEDIAN of the second half.

5.3 Aggregated Data

An aggregation process must produce one aggregated result, denoted as mandatory feature "Aggregate Data" in the feature diagram. Aggregated data may have a set of features, which are explained as follows.

Push: In some survey DAP examples, sending aggregated data to another unit of the system is an activity of the aggregator immediately after the computation of aggregation. This is considered as an active step of the aggregation process, and is represented by the feature "push". For example, in the group layer aggregation of VigilNet [29], each node sends the aggregated data to its leading node actively. An aggregation process without the "push" feature leaves the aggregate results in the main memory, and it is other processes' responsibility to fetch the results.

The aggregated data may be "pushed" into permanent storage, such as in [32] and [11]. The stored aggregated data may be required to be durable, which means that the aggregated data must survive potential system failures. Therefore, "**durable**" is considered as an optional sub-feature of the "push" feature.

Shared: Similar to raw data, the aggregated data has an optional "shared" feature too, to represent the characteristic of some of the surveyed DAPs that the aggregated data may be shared by other concurrent processes in the system. For instance, the aggregated results of one process may serve as the raw data inputs of another aggregation process, creating a hierarchy of aggregation [25], [29]. The results of aggregation may also be accessed by a non-aggregation process, such as a control process [9].

Time-to-live: The "time-to-live" feature regulates how long the aggregated data should be preserved in the aggregator. For instance, Aurora system [25] can be configured to guarantee that the aggregated data are available for other processes, such as an archiving process or another aggregate process, for a certain period of time. After this period, these data can be discarded or overwritten. We use the optional feature "time-to-live" to represent this characteristic.

Real-Time (AD): The aggregated data may be real-time, as required in some of the surveyed DAPs, if the validity of the data instance depends on whether its temporal consistency constraints are met. Therefore the "real-time" feature, which is named "Real-Time (AD)", is an optional feature of aggregated data in our taxonomy. The temporal consistency constraints on real-time aggregated data include two aspects, the absolute validity and relative validity, as explained in Section 3. "**Absolute validity interval**" and "**relative validity interval**" are two mandatory sub-features of the "Real-Time (AD)" feature.

Similar to raw data, the real-time feature of aggregated data has "**hard**", "**firm**" and "**soft**" as alternative sub-features. If the aggregated data are required to be hard real-time, they have to be ensured temporal consistent in order to avoid catastrophic consequences [32]. Compared with hard real-time data, firm real-time aggregated data are useless if they are not temporal consistent [29], while soft real-time aggregated data can still be used with less value (e.g., the aggregation in the remote server [11]).

5.4 Triggering Pattern

"Triggering pattern" refers to how the DAP is activated, which is a mandatory feature. We consider three types of triggering patterns for the activation of DAPs, represented by the alternative sub-features "**periodic**", "**sporadic**" and "**aperiodic**".

A periodic DAP is invoked according to a time schedule with a specified "Period". A sporadic DAP could be triggered by an external "event", or according to a time schedule, possibly with a "MinT" (Minimum inter-arrival Time) and/or "MaxT" (Maximum inter-arrival Time). An aperiodic DAP is activated by an external "event" without a constant period, MinT or MaxT. The event can be an aggregate command (e.g. an explicit aggregation query [28]) or a state change in the system [32].

5.5 Real-time (P)

Real-time applications, such as automotive systems [9] and industrial monitoring systems [11], require the data aggregation process to complete its work by a specified deadline. The process timeliness, named "**Real-Time (P)**", is considered as an optional feature of the DAP, and "**deadline**" is its mandatory sub-feature.

Aggregation processes may have different types of timeliness constraints, depending on the consequences of missing their deadlines. For a **soft** real-time DAP, a deadline miss will lead to a less valuable aggregated result [30]. For a **firm** real-time DAP [11], the aggregated result becomes useless if the deadline is missed. If a **hard** real-time DAP misses its deadline, the aggregated result is not only useless, but hazardous [9], [10]. "Hard", "firm" and "soft" are alternative sub-features of the timeliness feature.

We must emphasize the difference between timeliness ("Real-Time (P)") and real-time features of data ("Real-Time (RD)" and "Real-Time (AD)"), although both of them appear to be classified into hard, firm and soft real-time. Timeliness is a feature of the aggregation process, with respect to meeting its deadline. It specifies when the process must produce the aggregated data and release the system resources for other processes. As for real-time features of data, the validity intervals specify when the data become outdated, while the level of strictness with respect to temporal consistency decides whether outdated data could be used.

To meet the desired real-time strictness level of the data, the DAP may need to meet certain timeliness requirements, which will be discussed further in Section 6.

6 Design Rules and Heuristics

In the previous section we have introduced our taxonomy that encompasses the important features of a DAP. In this section, we formulate a set of design rules and heuristics, following the design implications imposed by the features. The design rules are the axioms that should be applied during the design. Violating the rules will result in infeasible feature combinations. Design heuristics, on the other hand, suggest that certain mechanisms may be needed, either to implement the selected features, or to mitigate the impact of the selected features.

6.1 Design Rules

The real-time features of data and process are commonly desired features of DAPs among real-time applications. Among these features there exist dependencies, which should be respected when one is selecting and combining these features. In this subsection we analyze the dependencies among the real-time data features (the "Real-Time (RD)" and "Real-Time (AD)" features in the taxonomy) and the timeliness feature (the "Real-Time (P)" feature) of the aggregation process itself. Based on the analysis we formulate three design rules to eliminate the infeasible combinations.

As already introduced, real-time data can be classified as hard, firm or soft real-time according to the strictness w.r.t. the temporal consistency. The hard real-time feature imposes strongest constraints and represents highest level of strictness, while the soft real-time feature represents the lowest level of strictness. From the raw data to the aggregated data, the level of strictness can only decrease or remain the same. This is because the validity of aggregated data depends on the validity of raw data. Since the hard real-time aggregated data have to be both absolute valid and relative valid, which requires all involved raw data to be absolute valid, the raw data have to be hard real-time too. If the raw data is soft real-time, which indicates that outdated raw data may occur, the temporal consistency of the aggregated data cannot be guaranteed. Therefore, we get the following rule:

Rule 1: **The real-time strictness level of the raw data must be higher than or equal to the real-time strictness level of the aggregated data**.

The timeliness of the entire data aggregation process has an impact on meeting the strictness level of the aggregated data, since the validity of the aggregated data depends on the interval between the time when raw data are collected, and the time when the aggregated data are produced. If the aggregated data are required to be hard real-time, the DAP also has to be hard real-time. If the timeliness of the DAP is soft, the calculation may miss its deadline and produce an outdated aggregated result. If we consider the "hard", "firm" and "soft" features of the DAP as levels of strictness w.r.t. timeliness, this rule is formulated as follows:

Rule 2: **The strictness level w.r.t. timeliness of the entire DAP must be higher than or equal to the real-time strictness level of the aggregated data**.

The fact that both the raw and aggregated data may be shared by multiple processes imposes further consideration on the real-time strictness of the shared data. If the raw data or the aggregated data are shared by several processes, and the requirements of these processes impose different real-time strictness, then the real-time constraint of this data is in accordance with the highest strictness required by these processes. For example, the raw data of an aggregate process happens to be the input of a control process that demands the input to be hard real-time. Even though the aggregation process can tolerate outdated raw data, the real-time strictness level of the data must be hard. Otherwise the data for the control process may be outdated and lead to catastrophic consequences. Hence we formulate the following rule:

Rule 3: **The real-time strictness of the raw/aggregated data must meet the highest real-time strictness level imposed by all processes that share the data**.

These rules should be applied when the application designer analyzes the features derived from the requirement specification. Consider a process aggregating data from three sensors and providing its aggregated data to a hard real-time control process. The specification of the aggregation process may allow outdated raw data, i.e., soft real-time raw data, and tolerate occasional deadline miss. However, since the control process requires its inputs (the aggregated data) to be hard real-time, both the raw data from the sensors and the DAP have to be hard real-time as well.

6.2 Design Heuristics

Accomplishing the design of a DAP involves the design of appropriate supporting run-time mechanisms. These mechanisms either achieve the selected features of the DAP, or mitigate the impact of the selected features in order to ensure other properties of the system. Such properties could be, for instance, the logical data consistency characterized by the ACID properties of the processes. In this subsection we introduce a set of design heuristics, which are suggestions of mechanisms that could be implemented in order to enforce certain features and system properties. The heuristics are organized as suggested mechanisms as follows.

Synchronization for "pull" and "push" features: Pulling raw data from a data source may involve locating the data source, selecting the data and shipping data into the aggregator. Pushing aggregated data may involve locating the receiver and transmitting the data. These activities introduce higher risks of delayed and missing data that may breach the temporal and logical data consistency. Overheads in time and computation resource are also introduced, which are impacting factors of the overall timeliness of the process. When designing for such systems, one may consider developing a synchronization protocol to mitigate such impacts and ensure the consistency of the data.

Load shedding for "sheddable" feature combined with real-time features: Situations could occur when the DAP is not able to meet the real-time constraints, due to, for example, system overload. If the raw data are sheddable, one may consider implementing the load shedding mechanism [25], which allows raw data instances to be discarded systematically.

Approximation for "sheddable" feature combined with real-time features: An alternative mechanism for sheddable raw data is to implement approximation techniques. For example, Deshpande et al. introduce an approximation technique into sensor network to improve the efficiency of aggregation [35]. Instead of reading data from all sensors, the DAP only collects raw data from some of the sensors that fulfill a probabilistic model.

Concurrency control for "shared" feature: An implication of shared data is the concern of logical data consistency, which is a common consideration from concurrent data access. A certain form of concurrency control needs to be implemented to achieve a desired level of consistency. For example, the aggregate process may achieve full isolation from other processes, i.e., they can only see the aggregated result when the DAP completes, using serializable concurrency control [36]. To improve performance or timeliness, one may choose a less stringent concurrency control that allows other processes to access the sub-aggregate results of the DAP, which may lead to a less accurate final result. Without any concurrency control, the aggregation process may produce incorrect results using inconsistent data [37].

Logging and recovery for "durable" feature: In order to ensure the "durable" aggregated data, logging and backward failure recovery techniques, which are commonly used to achieve durability in data management systems, may be applied to the DAP. For example, the operations on the aggregated data are logged immediately, and the actual changes are written into the storage periodically.

Filtering for "duplicate sensitive" aggregate functions: Using a duplicate sensitive aggregate function indicates a higher risk of inconsistency caused by duplicated values sent to aggregator. A filtering mechanism may be implemented to identify the duplicates and filter them away.

Caching for "lossy" aggregate functions: Lossy aggregate functions disallow the reconstruction of raw data from the aggregated data. However, raw data may be needed to redo all changes when errors occur, in order to ensure the atomicity of a process. A caching mechanism may be implemented for the DAP as a solution, that raw data instances are cached in the aggregator until the process completes.

Decomposition of aggregation for "progressive" aggregate functions: The implication of using a progressive aggregate function is that one may decompose the entire aggregation into sub-aggregates. Computing the sub-aggregates in parallel may benefit the performance of the entire DAP. Another useful application of the decomposition is error handling, especially when it is combined with a caching mechanism. Consider an aggregate process fetching data from several sensors. The process can perform aggregation upon the arrival of each sensor data and cache the sub-aggregate result so far. If an error occurs during the fetching of next sensor, the process can return the cached sub-aggregate result as an approximation [5], or only restart the fetching of the failed sensor, instead of restarting the whole process.

Buffers for raw data and aggregated data: Raw data arrive in the aggregator with their "MINT", which could be different from the aggregation interval imposed by the"triggering pattern" of the process. Buffers may be necessary to keep the raw data available for aggregation. Buffers may also be necessary for the aggregated data, since the aggregated data are generated according to the "triggering pattern", and must be available for a specified period defined by the "time-to-live" feature. Buffer management is crucial for the accuracy of aggregation as well as the resource utilization. For instance, circular buffer is a common mechanism in embedded systems for keeping data in limited memory. When the buffer is full, the program will just overwrite the old content with new data from the beginning. With the features presented in our taxonomy, one may calculate buffer size based on worst-case scenarios for non sheddable data, or suffice buffer size for sheddable data, given the size of each data instance.

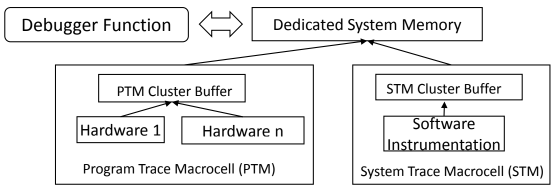


Fig. 8. General architecture of the Hardware Assisted Trace system

7 EVALUATION: AN INDUSTRIAL CASE STUDY

In this section we evaluate the usefulness of our taxonomy in the design of a data aggregation application. Prior to the case study we have implemented a tool called DAPComposer (Data Aggregation Process Composer), shown in Fig. 9. The tool provides a graphical user interface for designers to create DAPs, by selecting and arranging the features in the diagram. Rules of mandatory, optional and alternative features are implemented. The mandatory features are always enabled, while optional and alternative features can be enabled/disabled by double-clicking the features. Annotations can be added to the selected features, such as the name of the data, or the actual value of the timing properties. It can also hide disabled features to provide a cleaner representation. Constraints can be typed as rules by users and saved in a rule base. The tool then validates the design against the specified rules. Although to the date only primitive constraints intrinsic to the feature model are checked by DAPComposer, we plan to mature the tool with more sophistic analysis capabilities, such as timing analysis, in the next version.

This evaluation is conducted on an industrial project, the Hardware Assisted Trace (HAT) [38] framework, together with its proposers from Ericsson. HAT, as shown in Fig. 8, is a framework for debugging functional errors in an embedded system. In this framework, a debugger function runs in the same system as the debugged program, and collects both hardware and software run-time traces continuously. Together with the engineers we have analyzed the aggregation processes in their current design. At a lower level, a Program Trace Macrocell (PTM) aggregation process aggregates traces from hardware. These aggregated PTM traces, together with software instrumentation traces from the System Trace Macrocell (STM), are then aggregated by a higher level ApplicationTrace aggregation process, to create an informative trace for the debugged application.

We have analyzed the features of the PTM aggregation process and the ApplicationTrace aggregation process in HAT based on our taxonomy. The features of the PTM aggregation process are presented in Fig. 10. Triggered by computing events, this process pulls raw data from the local buffer of the hardware, and aggregates them using an encoding function to form an aggregated trace into the PTM cluster buffer. The raw data are considered sheddable, since they are generated frequently, and each aggregation pulls only the data in the local buffer at the time of the triggering event. The aggregated PTM and STM traces then serve as part of the raw data of the ApplicationTrace aggregation process, which is shown in Fig. 11. The ApplicationTrace process is triggered sporadically with a minimum inter-arrival time, and aggregates its raw data using an analytical function. The raw data of the ApplicationTrace should not be sheddable so that all aggregated traces are captured.

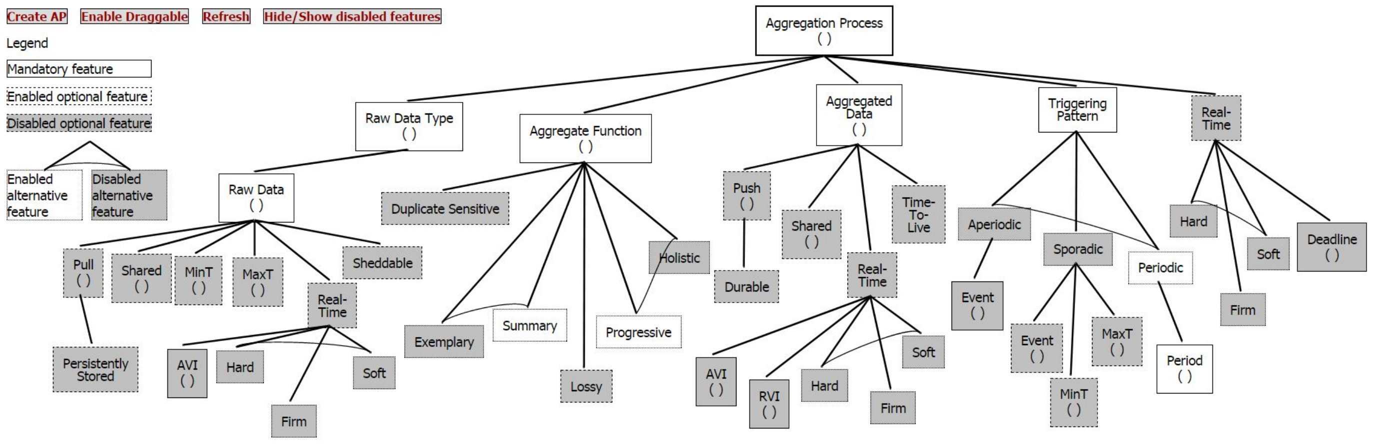


Fig. 9. Interface of DAPComposer

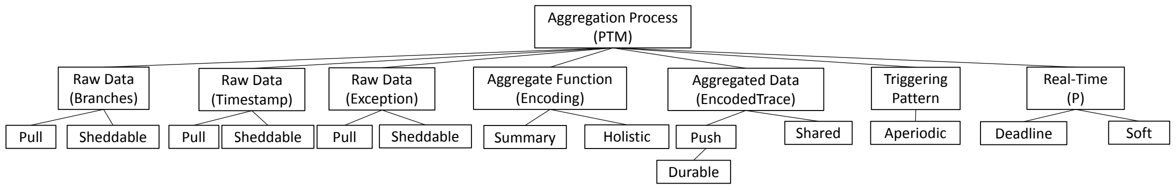


Fig. 10. The aggregation process in the PTM

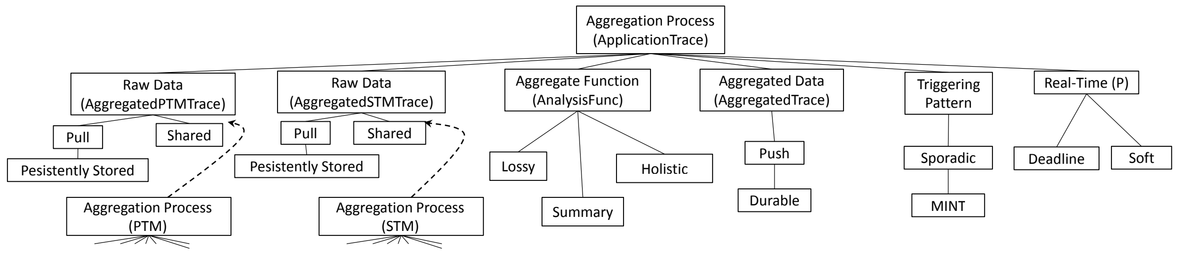

Aggregation Process (ApplicationTrace)
Raw Data (AggregatedPTMTrace)
Raw Data (AggregatedSTMTrace)
Aggregate Function (AnalysisFunc)
Aggregated Data (AggregatedTrace)
Triggering Pattern
Real-Time (P)
Pull
Shared
Pesistently Stored
Pull
Shared
Pesistently Stored
Lossy
Summary
Holistic
Push
Durable
Sporadic
MINT
Deadline
Soft
Aggregation Process (PTM)
Aggregation Process (STM)

Fig. 11. The aggregation processes in the investigated HAT system

7.1 Problem identified in the HAT design

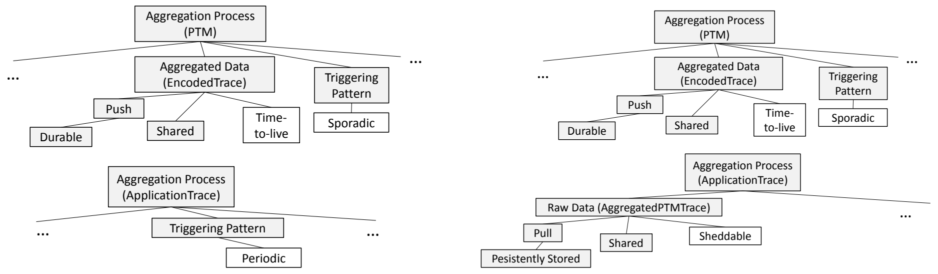
With the diagrams showing the features of the aggregation processes, the engineers could immediately identify a problem in the PTM buffer management. In the current design, the buffer size is decided by both the hardware platform and the designer's experiences. The problem is that, the data in the buffer may be overwritten before they are aggregated. This problem has been observed on Ericsson's implemented system, and awaits a solution. However, if the taxonomy would have been applied on the system design, this problem could have been identified before it was propagated to implementation.

This problem arises due to the lack of a holistic consideration on the PTM aggregation process and the ApplicationTrace aggregation process at design time. Triggered by aperiodic external events, the PTM process could produce a large number of traces within a short period and fill up the PTM buffer. The ApplicationTrace process, on the other hand, is triggered with a minimum inter-arrival time, and consumes the PTM traces as unsheddable raw data. When the inter-arrival time of the PTM triggering events is shorter than the MINT of the ApplicationTrace process, the PTM traces in the buffer may be overwritten before they could be aggregated by the ApplicationTrace process.

7.2 Solutions

Providing a larger buffer could be a choice to mitigate this problem. However, a larger provision might either still fail to meet the buffer consumption in some rare cases, or become a loss of resource due to pessimism. Considering the resource-constrained nature of the system, a better way is to derive the necessary buffer size at design time, given the size of each data entry. Based on our taxonomy, we and Ericsson engineers have come up with two alternative design solutions to fix this problem. Both solutions reuse most of the features in the current design.

Solution 1: To be able to derive the worst-case buffer size, one solution is to ensure more predictable behaviors of the aggregation processes, by adjusting the following features in the diagram (see Fig. 12a): (i) Instead of selecting the "aperiodic" feature, the PTM process should select "sporadic", with a defined MINT; and, (ii) the "sporadic" feature of the ApplicationTrace process should be replaced by "periodic", so that the frequency of consuming the aggregated PTM traces can be determined. These changes of the features entail introducing extra real-time mechanisms into the current design, such as an admission control to ensure the MINT and a scheduler to schedule the processes. In addition, a "time-to-live" feature, whose value equals to the period


Aggregation Process (PTM)
Aggregated Data (EncodedTrace)
Triggering Pattern
Push
Durable
Shared
Time-to-live
Sporadic
Aggregation Process (ApplicationTrace)
Triggering Pattern
Periodic
Raw Data (AggregatedPTMTrace)
Pull
Pesistently Stored
Sheddable

(a) Solution 1 (b) Solution 2

Fig. 12. Illustration of Solution 1 and Solution 2. Unchanged features from the current design are marked in gray.

of the ApplicationTrace process, should be added to the PTM process. To achieve this, a mechanism that prevents traces from being overwritten before the "time-to-live" value needs to be introduced. These new features allow the designer to analyze the worst-case production and consumption of the aggregated PTM traces, and therefore derive the worst-case buffer size for the system using the actual values of the features. This solution also ensures that all PTM traces are aggregated by the ApplicationTrace process. Features reused from the current design are either marked in gray color, or omitted for readability.

Solution 2: An alternative to derive the exact buffer size is to allow overwriting in a controlled manner, as illustrated in Fig. 12b. On one hand, as in Solution 1, we suggest to replace the "aperiodic" feature of the PTM process with "sporadic", so that the worst-case buffer size for PTM trace production can be determined. On the other hand, the triggering pattern of the ApplicationTrace process remains unchanged ("sporadic"). However, a "sheddable" feature from the taxonomy is added to the raw data of the ApplicationTrace process, while a "time-to-live" is added to the PTM process. A shedding mechanism needs to be introduced, which in this case could be a logic in the buffer management that allows overwriting traces older than the "time-to-live" value. For instance, the designer may decide that the PTM traces produced 10 milliseconds ago are not valuable for the ApplicationTrace. When a PTM trace is older than 10 milliseconds, it might be overwritten even though it has not been aggregated. With the knowledge of the worst-case production of the PTM traces, and the "time-to-live" value of each trace, the designer is able to derive the needed buffer size.

Both solutions guarantee bounded buffers, while they require just a few features to be changed, and mechanisms introduced accordingly in the current design. Compared with Solution 2, which could lose traces, Solution 1 ensures all generated traces to be aggregated. However, to enforce a periodic triggering pattern as suggested in Solution 1, more efforts are required to provide real-time support, such as a real-time operating system.

7.3 Comparison with other taxonomies

Analysis based on other taxonomies could not easily identify the aforementioned bottleneck, since they characterize other aspects of data aggregation. The taxonomies proposed by Madden et al. [5] and Gray et al. [2] can only be applied to describe the aggregate functions of HAT, which are also captured by our taxonomy. The taxonomies of Solis et al. [7], Makhloufi et al. [6], and Rajagopalan [15], do not support modeling of data properties. Although Fasolo et al. [4] have considered data representations, aggregate functions and aggregation protocols, their taxonomy is defined at a much coarser level and does not allow for the analysis on such detailed data and process behaviors as the aforementioned bottleneck.

7.4 Summary

The engineers in the evaluation acknowledge that our taxonomy bridges the gap between the properties of data and the properties of the process, which has not been elaborated by other taxonomies. Our taxonomy enhances the understanding of the system by structuring the common and variable features of data aggregation processes. By applying analysis based on our taxonomy, design flaws can be identified and fixed prior to implementation, which improves the quality of the system and saves money. Design solutions can be constructed by composing reusable features, and reasoned about based on the taxonomy, which contributes to a reduced design space. Due to these benefits, the engineers see great value in a potential design tool for data aggregation applications based on our taxonomy.

8 Discussion

The proposed taxonomy brings new lights on the understanding of the complexity of data aggregation in general. With a structured, feature-based organization, our taxonomy can be viewed as a common framework in which one can study the implications of the features, as well as dependencies between the features and the processes. Henceforth, one can reason about how the selection of one feature will influence other features, or even other processes. From an engineering perspective, as demonstrated by the HAT case study, applying our taxonomy to analysis can help designers identify flaws prior to implementation, and find new design solutions.

One direct usage of the dependencies between features is to eliminate infeasible feature combinations in the design. The rules regarding real-time strictness levels of data and process timeliness in Section 6 are one example. Although they appear straightforward and general, these rules regulate the qualitative relationships between real-time and timeliness features, and thus reduce the design space. For more accurate analysis, such as the derivation of buffer size in Section 7, values of quantitative features such as deadline and period should be involved into the calculation.

Conflicts may occur between real-time features and the other features. For instance, a DAP with durable aggregated data will ideally store each result into permanent storage, introducing frequent disk I/O which is time consuming and unpredictable. This may contradict the requirement of bounded computation time imposed by the timeliness feature. Such conflicts can be generalized as conflicts between logical data consistency, temporal data consistency and timeliness. Features such as "durable" and "shared" data are closely related to logical data consistency. The suggested mechanisms "logging and recovery" and "concurrency control" are common means to achieve durability and isolation respectively. They all have impacts on the temporal data consistency and timeliness. In such cases, a simple rule to detect potential conflicts is not possible. Neither is it possible to formulate a rule that resolves the conflicts that occur. We believe the conflict detection, as well as the trade-offs among conflicting features, must come from careful analysis that relies on the selected features of a particular configuration with their values in the real case. Advanced analysis techniques, such as model-checking [39], can be applied on the configuration to verify whether the desired properties hold, and guide the trade-offs.

9 Conclusion

In this paper, we have investigated the characteristics of data aggregation processes in a variety of applications, and provided a taxonomy of the DAPs, with a particular focus on the real-time properties. Our taxonomy is presented as a feature diagram, in which the common and variable characteristics are modeled as features. The taxonomy provides a comprehensive view of data aggregation processes for the designers, and allows the design of a DAP to be achieved via the selection of desired features and the combination of the selected features.

Based on the implications of the features in the taxonomy, we have introduced three rules that should be followed during the design of DAPs. These rules eliminate some of the infeasible combinations of features during the design. We have also proposed a set of design heuristics, which help the designer to decide the necessary mechanisms for achieving the selected features and other system properties. The usefulness of the taxonomy has been demonstrated by an industrial case study. Flaws can be identified at design time, and solutions can be proposed at design level, by applying the taxonomy to the analysis.

Our taxonomy can be viewed as a framework for analyzing the dependencies between features and aggregation processes. For some potential conflicts among the desired features, we have highlighted that trade-offs must be decided based on careful analysis. More advanced analysis techniques are needed for reasoning about the conflicts among selected features, as well as the possible resolutions. In our future work, we plan to apply advanced analysis techniques, such as model-checking, to facilitate the trade-offs among features during the design of data aggregation processes.

Acknowledgment

This work is funded by the Knowledge Foundation of Sweden (KK-stiftelsen) within the DAGGERS project. We acknowledge Alf Larsson and Carlo Vitucci for their help in the case study.

References

[1] I. J. Rudas, E. Pap, and J. Fodor, "Information aggregation in intelligent systems: An application oriented approach," *Knowledge-Based Systems*, vol. 38, pp. 3–13, 2013.

[2] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 29–53, 1997.

[3] N. Iftikhar, "Integration, aggregation and exchange of farming device data: A high level perspective," in *Proceedings of the 2nd International Conference on the Applications of Digital Information and Web Technologies*, 2009, pp. 14–19.

[4] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *Wireless Communications, IEEE*, vol. 14, no. 2, pp. 70–87, 2007.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[6] R. Makhloufi, G. Doyen, G. Bonnet, and D. Gaïti, "A survey and performance evaluation of decentralized aggregation schemes for autonomic management," *International Journal of Network Management*, vol. 24, no. 6, pp. 469–498, 2014.

[7] I. Solis and K. Obraczka, "In-network aggregation trade-offs for data collection in wireless sensor networks," *International Journal of Sensor Networks*, vol. 1, no. 3-4, pp. 200–212, 2006.

[8] H. Alzaid, E. Foo, J. M. G. Nieto, and D. Park, "A taxonomy of secure data aggregation in wireless sensor networks," *International Journal of Communication Networks and Distributed Systems*, vol. 8, no. 1-2, pp. 101–148, 2012.

[9] G. Goud, N. Sharma, K. Ramamritham, and S. Malewar, "Efficient real-time support for automotive applications: A case study," in *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006, pp. 335–341.

[10] K. Bür, P. Omiyi, and Y. Yang, "Wireless sensor and actuator networks: Enabling the nervous system of the active aircraft," *Communications Magazine, IEEE*, vol. 48, no. 7, pp. 118–125, 2010.

[11] A. N. Lee and J. L. M. Lastra, "Data aggregation at field device level for industrial ambient monitoring using web services," in *Proceedings of the 9th IEEE International Conference on Industrial Informatics*. IEEE, 2011, pp. 491–496.

[12] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-021, 1990. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231

[13] R. Mesiar, A. Kolesárová, T. Calvo, and M. Komorníková, "A review of aggregation functions," in *Fuzzy Sets and Their Extensions: Representation, Aggregation and Models*. Springer Berlin Heidelberg, 2008, vol. 220, pp. 121–144.

[14] J.-L. Marichal, *Aggregation Functions for Decision Making*. ISTE, 2010, pp. 673–721.

[15] R. Rajagopalan and P. Varshney, "Data-aggregation techniques in sensor networks: A survey," *Communications Surveys Tutorials, IEEE*, vol. 8, no. 4, pp. 48–63, 2006.

[16] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[17] X. Song and J. Liu, "How well can data temporal consistency be maintained?" in *Proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design (CACSD)*, 1992, pp. 275–284.

[18] K. Czarnecki and E. Ulrich, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[19] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "Featureide: An extensible framework for feature-oriented software development," *Sci. Comput. Program.*, vol. 79, pp. 70–85, 2014.

[20] S. Chaudhuri and U. Dayal, "An overview of data warehousing and olap technology," *SIGMOD Rec.*, vol. 26, no. 1, pp. 65–74, 1997.

[21] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, 1st ed. Morgan Kaufmann Publishers Inc., 1992.

[22] I. F. V. Lopez, R. T. Snodgrass, and B. Moon, "Spatiotemporal aggregate computation: A survey," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 2, pp. 271–286, 2005.

[23] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy, "Answering queries with aggregation using views," in *Proceedings of the 22th International Conference on Very Large Data Bases*, 1996, pp. 318–329.

[24] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online aggregation," *SIGMOD Rec.*, vol. 26, no. 2, pp. 171–182, 1997.

[25] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A new model and architecture for data stream management," *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[26] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006, pp. 623–634.

[27] M. Oyamada, H. Kawashima, and H. Kitagawa, “Data stream processing with concurrency control,” *SIGAPP Appl. Comput. Rev.*, vol. 13, no. 2, pp. 54–65, 2013.

[28] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “Tinydb: An acquisitional query processing system for sensor networks,” *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[29] T. He, L. Gu, L. Luo, T. Yan, J. Stankovic, and S. Son, “An overview of data aggregation architecture for real-time tracking with sensor networks,” in *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, 2006, pp. 8 pp.–.

[30] B. Defude, T. Delot, S. Ilarri, J.-L. Zechinelli, and N. Cenerario, “Data aggregation in vanets: The vespa approach,” in *Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, 2008, pp. 13:1–13:6.

[31] B. Arai, G. Das, D. Gunopulos, and V. Kalogeraki, “Approximating aggregation queries in peer-to-peer networks,” in *Proceedings of the 22nd International Conference on*, 2006, pp. 42–42.

[32] J. Baulier, S. Blott, H. F. Korth, and A. Silberschatz, “A database system for real-time event aggregation in telecommunication,” in *Proceedings of the 24rd International Conference on Very Large Data Bases*, 1998, pp. 680–684.

[33] A. Bar, P. Casas, L. Golab, and A. Finamore, “Dbstream: An online aggregation, filtering and processing system for network traffic monitoring,” in *Proceedings of the 2014 International Wireless Communications and Mobile Computing Conference*, 2014, pp. 611–616.

[34] L. Golab, T. Johnson, J. S. Seidel, and V. Shkapenyuk, “Stream warehousing with datadepot,” in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009, pp. 847–854.

[35] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, “Model-driven data acquisition in sensor networks,” in *Proceedings of the 13th International Conference on Very Large Data Bases*, 2004, pp. 588–599.

[36] I. Botan, P. M. Fischer, D. Kossmann, and N. Tatbul, “Transactional stream processing,” in *Proceedings of the 15th International Conference on Extending Database Technology*, 2012, pp. 204–215.

[37] L. Gürgen, C. Roncancio, C. Labbé, and V. Olive, “Transactional issues in sensor data management,” in *Proceedings of the 3rd Workshop on Data Management for Sensor Networks*.

[38] C. Vitucci and A. Larsson, “Hat, hardware assisted trace: Performance oriented trace & debug system,” in *Proceedings of 26th International Conference on Software & Systems Engineering and their Applications*, 2015.

[39] S. Cai, B. Gallina, D. Nyström, and C. Seceleanu, “Trading-off data consistency for timeliness in real-time database systems,” in *Proceedings of 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 13–16.

[40] B. Babcock, M. Datar, and R. Motwani, “Load shedding for aggregation queries over data streams,” in *Proceedings of the 20th International Conference on Data Engineering*, 2004, pp. 350–361.

[41] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, “Form: A feature-oriented reuse method with domain-specific reference architectures,” *Annals of Software Engineering*, vol. 5, no. 1, pp. 143–168, 1998.

[42] K. Czarnecki, S. Helsen, and U. Eisenecker, “Formalizing cardinality-based feature models and their specialization,” *Software process: Improvement and practice*, vol. 10, no. 1, pp. 7–29, 2005.

[43] S. Eichler, C. Merkle, and M. Strassberger, “Data aggregation system for distributing inter-vehicle warning messages,” in *Proceedings of the 39th Annual IEEE Conference on Local Computer Networks*, 2006, pp. 543–544.

[44] H. Schweppe, A. Zimmermann, and D. Grill, “Flexible on-board stream processing for automotive sensor data,” *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 1, pp. 81–92, 2010.

[45] J. Kulik, W. Heinzelman, and H. Balakrishnan, “Negotiation-based protocols for disseminating information in wireless sensor networks,” *Wirel. Netw.*, vol. 8, no. 2/3, pp. 169–185, 2002.

[46] G. Graefe, “Query evaluation techniques for large databases,” *ACM Comput. Surv.*, vol. 25, no. 2, pp. 73–169, 1993.