Noname manuscript No.
(will be inserted by the editor)

Data Aggregation Processes: A Survey, A Taxonomy, and Design Guidelines

Simin Cai · Barbara Gallina · Dag Nyström · Cristina Seceleanu

Received: date / Accepted: date

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, many with timing constraints. Understanding the common and variable features of data aggregation processes, especially their implications to the time-related properties, is key to improving the quality of the designed system and reduce design effort. In this paper, we present a survey of data aggregation processes in a variety of application domains from literature. We investigate their common and variable features, which serves as the basis of our previously proposed taxonomy called DAGGTAX. By studying the implications of the DAGGTAX features, we formulate a set of constraints to be satisfied during design, which helps to check the correctness of the specifications and reduce the design space. We also provide a set of design heuristics that could help designers to decide the appropriate mechanisms for achieving the selected features. We apply DAGGTAX on industrial case studies, showing that DAGGTAX not only strengthens the understanding, but also serves as the foundation of a design tool which facilitates the model-driven design of data aggregation processes.

Keywords Data aggregation taxonomy · Real-time data management · Data modeling

1 Introduction

In modern information systems, data aggregation has long been adopted for data processing and management in order to discover unusual patterns and infer information [25], to save storage space [29], or to reduce bandwidth and energy costs [20]. Amid the era of cloud computing and Internet of Things (IoT), the application of data aggregation is becoming increasingly common and important, when enormous amounts

Mälardalen Real-Time Research Centre, Mälardalen University,
Västerås, Sweden
{simin.cai,barbara.gallina,
dag.nystrom,cristina.seceleanu}@mdh.se

of data are continuously collected from ubiquitous devices and services, and further analyzed. As an example, a surveillance application monitors a home by aggregating data from a number of sensors and cameras. The aggregated surveillance data of individual homes could then be aggregated again in the cloud to analyze the security of the area. In this example, data aggregation serves as a pillar of the application's workflow, and directly impacts the quality of the software system.

Within such systems, different aggregations may have various requirements to be satisfied by the design. For instance, while one aggregation receives data passively from a data source, another aggregation must actively collect data from a database which is shared concurrently by other processes. This heterogeneity increases the difficulty in designing a suitable solution with multiple aggregations. In addition, many applications such as automotive systems [22], avionic systems [8] and industrial automation [39] have timing constraints on both the data and the aggregation processes themselves. The validity of data depends on the time when they are collected and accessed, and the correctness of a process depends on whether it completes on time. These real-time constraints also add to the complexity of data aggregation design.

In this paper, we focus on the design support for data aggregation processes (or DAP for short), which are defined as the processes of producing synthesized forms from multiple data items [52]. The main constituents of a DAP include: the raw data as the source of aggregation, the aggregate function that performs computation on the raw data, and the aggregated data as the result of the aggregation. We consider a DAP as a sequence of three ordered steps, each possibly involving a series of activities on these constituents, as follows:

1. **Preparation of the raw data.** A DAP starts with preparing the raw data required for the aggregation, from the data source into the aggregation unit called the aggregator. This step may involve the locating, extraction, transportation, and normalization of raw data, if necessary.
2. **Aggregation of the raw data.** An aggregate function is applied by the aggregator that transforms the raw data into the aggregated data.
3. **Post-handling of the aggregated data.** The aggregated data may be further handled by the aggregator, for instance, saved into persistent storage or provided for other processes.

Several existing works [41,47,31] have addressed the issue of finding a "correct" aggregate function that serves the aggregation purpose of a given set of raw data, while few of them have focused on the design of DAP after the data and the aggregate function are decided, such that the desired constraints are met. In this paper, we aim to support design of an entire aggregation process such that the timing constraints can be satisfied. To achieve this, the designer must identify the time-related properties for the constituents of DAP, and features that can influence these properties, in a high level. Such features, ranging from functional features (such as data sharing) to extra-functional features (such as the strictness of timing constraints), are varying depending on different applications. Therefore, we have proposed a high-level taxonomy of data aggregation processes, called DAGGTAX (Data AGGregation TAXonomy) [10]. Presented as a feature diagram [33], DAGGTAX provides a comprehensive view of DAP for designers, by identifying its mandatory features as well as the optional ones. The usefulness of DAGGTAX has been demonstrated via two industrial case studies,

which have shown that DAGGTAX raises the awareness of design issues in DAP, and helps to reason about possible trade-offs between different design solutions.

This paper extends the work in [10] in three aspects. First, we present a survey of DAP in literature, which aims to investigate the common and variable features of DAP, and serves as an inspiration of DAGGTAX. The presented survey not only justifies the proposal of DAGGTAX, but also strengthens the understandings of DAP, especially its applications in various domains, for the community. In addition, in this paper, we strive to enlighten the implication of the features in DAGGTAX, and promote the reasoning of them during design time. Conflicts may arise among features during design, in that the existence of one feature may prohibit another one. In this case, trade-offs should be taken into consideration at design time, so that infeasible designs can be ruled out at an early stage. Therefore, we propose a set of design constraints and heuristics to aid the design trade-offs. The design constraints are axioms to be followed in order to achieve a correct design, while the heuristics, in a less formal presentation, provides design suggestions for realizing the specified DAP. Third, we provide a new case study, on a Brake-By-Wire (BBW) system, and demonstrate the usefulness of DAGGTAX in analyzing high-level timing specifications.

In brief, this paper extends [10] with the following contributions:

- a survey of DAP in literature, which serves as the basis of DAGGTAX;
- a set of design constraints and heuristics based on DAGGTAX.
- a new case study of analyzing a BBW system with DAGGTAX.

The remainder of the paper is organized as follows. In Section 2 we provide background information. Our survey is presented in Section 3, followed by DAGGTAX in Section 4. We propose the design rules and heuristics in Section 5, and present the case studies in Section 6. In Section 7, we discuss the existing taxonomies of data aggregation. Finally, we conclude the paper and outline the future work in Section 8.

2 Background

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

2.1 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 [9]. The property of completing the computation by a given deadline is referred to as *timeliness* [9]. The time interval between the activation and the completion of the task is named the response time. A real-time task can be classified as *hard*, *firm* or *soft* real-time, depending on the consequence of a deadline miss [9]. 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. Such an example 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* [9]. 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. Similarly, *MAXimum inter-arrival Time (MAXT)* 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 is 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* [57]. 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 (AVI)*. 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 (RVI)*.

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 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.

Let us illustrate the difference between timeliness and temporal data consistency via a DAP in an automotive system. We consider the braking system in a car which calculates the braking torque by aggregating the speed of wheels and the brake pressure. Timeliness requires that the braking torque must be generated by the DAP within a deadline of, for instance, 100ms. This deadline is hard, as violating it may lead to a crash. Temporal validity requires that the wheel speed must be sampled within a certain interval, for instance, 90ms. This interval is also hard, because using outdated speed data may also cause an accident, and hence must be avoided.

2.2 Feature Model and Feature Diagram

The notion of *feature* was first introduced by Kang et al. in the Feature-Oriented Domain Analysis (FODA) method [33], 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

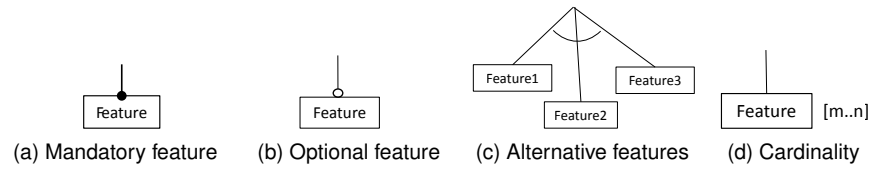


Fig. 1 Notations of a feature diagram

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 [15]. 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 obtained via a configuration process, selected from the feature model of its family.

A feature model is usually represented as a *feature diagram* [33], 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 [59], 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.

3 A Survey of Data Aggregation Processes

Based on scientific literature, in this section, we present a survey of application examples that implement data aggregation processes. We select these examples because, on one hand, DAP play an essential role in the solutions presented in this literature. On the other hand, these examples comprise applications from a wide variety of domains, which can help us to conclude the common and different characteristics of aggregation processes in general. We discuss the examples in three major categories. The first category, presented in Section 3.1, are general-purpose infrastructures that implement

aggregation as a basic service. The second category focuses on the type of emerging smart "X" applications (Section 3.2), in which data aggregation is commonly applied. The third category, which includes examples that develop data aggregation as ad hoc solutions suitable for the particular application scenarios, is presented in Section 3.3.

3.1 General-purpose Data Management 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) [60] 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, 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 data from disk into the main memory. An aggregate function is performed on the data 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 [24] 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 [42]. In a data warehouse, the stored data usually have many dimensions, and the aggregation may be performed on multiple dimensions [60]. Oftentimes, data need to be loaded from various data sources via the ETL (Extraction, Transformation, Loading) process, which extracts, validates and normalizes raw data before aggregation [60]. Therefore, the ETL process can be viewed as the preparation of raw data in a DAP in a high level.

The aggregated value may be returned to the query issuer directly, or may be stored persistently in the database. Alternatively, the aggregated values are cached in materialized views, so that other processes can make use of them [58]. It is common to store the aggregated values as materialized views in data warehouses since these results will be frequently used by analysis processes [60].

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 can be planned periodically, or issued on purpose, in order to refresh the materialized views that contains aggregated data [60].

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" [28], 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. 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 [1], 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 [37]. Oyamada et al. [48] 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" [20]. 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.

Madden et.al [43] 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. The aggregated results are cached by the nodes, and can be used for fault tolerance reasons, 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 [44], 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.

3.2 Smart "X", IoT and Big Data Applications

Data aggregation have been massively applied in the smart "X" domains, in the latest decade. Facilitated by the Internet of Things (IoT), big data analysis and cloud techniques, smart "X" addresses challenges and opportunities of a fully interconnected and integrated world, in which both computation powers and data are ubiquitous. Such examples include smart home [17], smart factory [40] and smart city [53].

Smart home applications improve the health and quality of life of the residents, and prevent emergencies in the home. Smart factory applications not only contribute to production and maintenance via optimized resource allocation and scheduling. Smart city applications assist to provide better services for citizens and avoid hazards. In spite of the various scope, these applications rely their decisions on analyzing the massive data collected from different sensors, monitors and software systems. These raw data, collected using various methods, are aggregated though a multi-layer aggregation system. Taking smart city as an example[40], raw data may be aggregated first in the community level, to form a report of the community. This aggregated data are then aggregated in district level, in the end the city level. Data may also be first aggregated in sub-domains, such as transportation, environment, etc., which in the end are aggregated for a comprehensive decision.

Due to the heterogeneous requirements of different sub-systems and sub-domains, the data aggregation processes in these applications have different characteristics. Data may be extracted from different sources, with particular efforts for transportation and transformation. Aggregated results may be exploited immediately in the local level, or be transmitted to the data center. Timing constraints are also different in

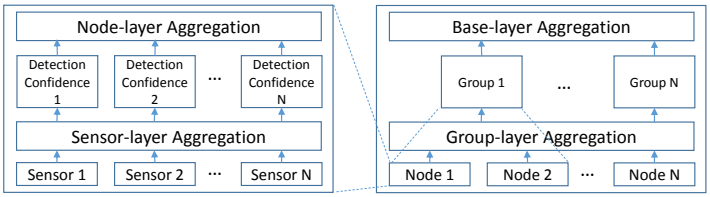


Fig. 2 Data Aggregation Architecture of VigilNet [27]

different scenarios. Decisions for timely reactions, such as handling of the falling of an elderly, or organizing the real-time traffic volumes in an exhibition, may require to use timely data, and meet strict deadlines of the aggregation process [36, 35], while historical analysis and long-term prognosis may impose less stringent requirements in terms of timing constraints. Various aggregate functions are applied to achieve the analysis goals. For scenarios with large amount of data, aggregate functions may even relax the accuracy and provide approximate results in order for better timeliness [62].

3.3 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 [27]. 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. 2. 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 [16] 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. [22] 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 systems [3]. 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. [6] 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 for other time-critical processes. When a new event arrives, it triggers the aggregate process to update the aggregate view. The event record is stored into a persistent data warehouse, which is not time-critical.

Bar et al. [5] 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 [8]. 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 [39]. Three types of aggregation are described, which are aggregation at the device level, aggregation in the control system, and aggregation in the 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 [29]. 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, aggrega-

tion 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 [21], 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.

3.4 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, 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 DAP, 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. Therefore, we present a taxonomy based on these characteristics as the systematic representation in the next section.

4 Our Proposed Taxonomy

Based on the characteristics revealed by our survey in Section 3, we have proposed a taxonomy of data aggregation processes called DAGGTAX, which first appeared in [10]. The taxonomy is shown in Fig. 3.

Table 1 Characteristics of Data Aggregation Processes in the Surveyed Applications

Example	Raw Data	Aggregate Function	Aggregated Data	Triggering Pattern	Deadline
relational disk-based DBMS/DW [58,42,12]	pulled from data sources; persistently stored; possibly shared	various functions	possibly durable; possibly shared	by events, or periodically	usually no
DSMS [1,48, 37]	pushed by data sources; possibly pushed periodically; not persistently stored; real-time; possibly shared; possibly shedded	various functions	pushed to receiver; possibly durable; possibly maintained for a period	by events, or periodically	depending on the application
WSN [43,44])	pulled from data sources; not persistently stored; possibly skipped	various functions	possibly maintained for a period; possibly durable; real-time; possibly shared	by events, or periodically	depending on the application
Smart "X" [17, 36,40,62,35, 53])	possibly pulled from data sources; possibly persistently stored; possibly skipped	various functions	possibly maintained for a period; possibly durable; possibly real-time; possibly shared	by events, sporadically or periodically	depending on the application
VigilNet [27], sensor layer	pushed by data sources; not persistently stored; real-time; pushed periodically	confidence function	pushed to receiver; not durable	periodically	hard
VigilNet [27], node layer	pushed by data sources; not persistently stored	average	pushed to receiver; not durable	by event	soft
VigilNet [27], group layer	pushed by data sources	ad hoc function	pushed to receiver; not durable	by event	soft
VigilNet [27], base layer	pushed by data sources; persistently stored	regression	shared	by event	soft
VESPA [16]	pushed by data sources	various functions	durable; shared	by events	soft
Goud et al. [22], EDR	pulled from data sources; pulled periodically; real-time; not persistently stored	various functions	not durable; real-time; shared	periodically	hard
Goud et al. [22], DDR	pulled from data sources; real-time; not persistently stored	various functions	durable; real-time	by events	hard
Arai et al. [3]	pulled from data sources; not persistently stored	various functions	possibly durable	by events	no
Baulier et al. [6]	pushed by data sources; persistently stored	various functions	real-time; not durable; shared	by events	hard
Bar et al. [5]	pushed by data sources; persistently stored; possibly real-time	various functions	durable	by events, or periodically	soft
Bür et al. [8]	pushed by data sources; not persistently stored; real-time;	various functions	not durable; real-time	periodically	hard
Lee et al. [39], device	pushed by data sources; real-time	various functions	pushed to receiver; not durable	by events, or periodically	soft
Lee et al. [39], control	pulled from data sources; persistently stored	various functions	possibly durable	periodically	soft
Lee et al. [39], monitoring	pulled from data sources; persistently stored	various functions	possibly durable	periodically	soft
Iftikhar [29]	pulled from data sources; persistently stored; stored for a period; possibly shared	various functions	durable; stored for a period; possibly shared	periodically	soft
DataDepot [21]	pulled from data sources; not persistently stored; possibly shared; real-time	various functions	durable; real-time	by events	depending on the application

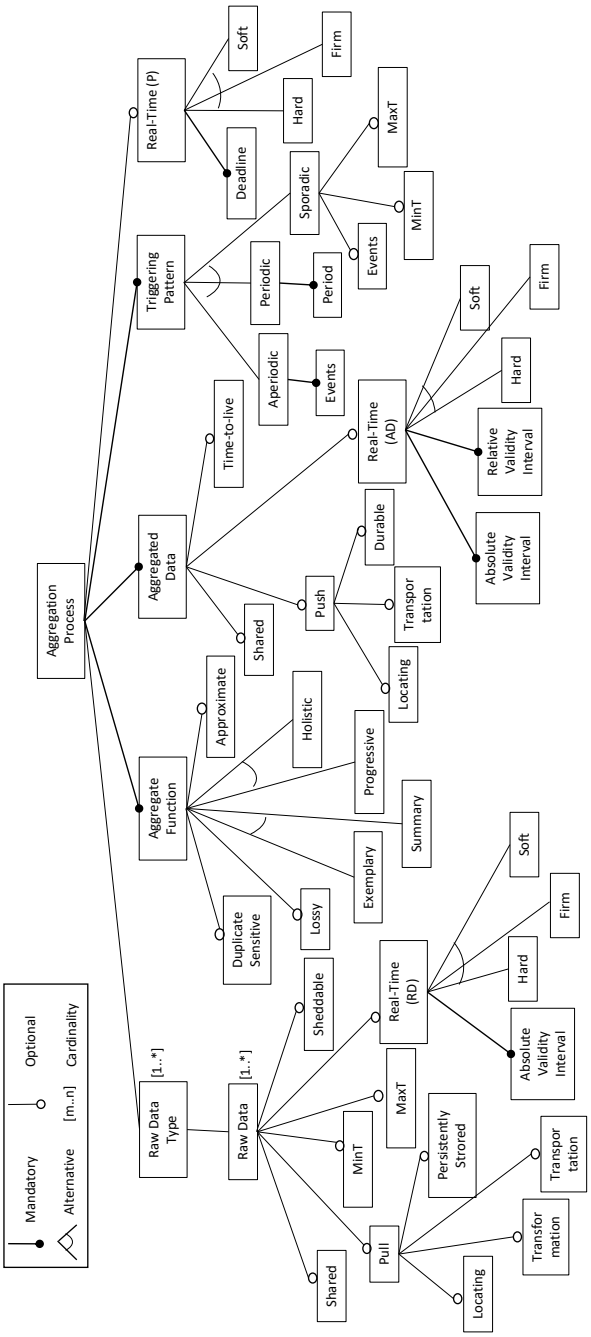


Fig. 3 The taxonomy of data aggregation processes

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. 3 from the top-level features under "Aggregation Process", including "Raw Data Type", "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. Sub-features of the top-level features are explained in a depth-first way.

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. 3. 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. 4a. 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 the data source. If raw data have the "pull" feature, pulling raw data actively from the data source is a necessary part of the DAP, 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. 4b). In this case, in our view the action of pushing data is the responsibility of another process outside of the DAP. From the DAP's perspective, the raw data are already prepared for aggregation.

We recognize several sub-features that are often involved in the pull of raw data. An optional sub-feature is "**Persistently Stored**", since raw data to be pulled from data source may be stored persistently in a non-volatile storage, such as a disk-based relational DBMS. The retrieval of persistent raw data involves locating the data in the storage and the necessary I/O. The optional sub-feature "**Locating**" represents the locating of the raw data, to which a selection condition could be applied. "**Transformation**" is an optional sub-feature regarding the necessity to transform the raw data into a normalized form. The optional sub-feature "**Transportation**" indicates the shipping of raw data from the data source. These aforementioned sub-features of "pull" all contribute to the overhead in time and thus can influence the timing properties.

Shared. Raw data of a DAP may be read or updated by other processes at the same time when they are read for aggregation [29]. The same raw data may be aggregated by several DAP, 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.

Fig. 4 Raw data acquisition schemes

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 [43], 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 [1]. 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 DAP 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)") and the process ("Real-Time (P)").

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 2, hard real-time data (such as sensor data from a field device [39]) and firm real-time data (such as surveillance data [27]) 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 [27]).

MINT and ***MAXT***. 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 VigilNet [27], 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. Similarly a raw data may have a MAXimum inter-arrival Time (MAXT). We consider "MINT" and "MAXT" optional features of the raw data.

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. [43] and Fasolo et al. [20]. 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 sensi-

tive 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 [43], an aggregate function is either “exemplary” or “summary”, which are 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 .

Lossy. An aggregate function is “lossy”, if the raw data cannot be reconstructed from the aggregated data alone [20]. For example, SUM, which computes the summation of a set of raw data, 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.

Approximate. An aggregate function may generate an approximation, instead of an exact result of an aggregation, especially when the raw data set is large, and the time or computation resource is limited. We refer to such an optional feature as “Approximate”. For instance, Yun et al. [62] have applied approximate aggregate functions to generate estimations used for the final aggregated value. The time spent by the DAP is reduced thanks to the approximation.

Holistic or Progressive. Researchers have classified aggregate functions based on whether or not the computation of aggregation can be decomposed into sub-aggregations. Using this differentiating criteria, Lenz et al. [41] classify aggregate functions as summarizable or non-summaraizable, and checks the semantic correctness (summarizability) of the aggregation. Using the same criteria, Gray et al. [25] and Madden et.al [43] classify them as holistic and non-holistic, when they analyze the operational performance of aggregation with respect to response time. In this paper, we inherit the classification terms from the latter (Gray et al. [25] and Madden et al. [43]), since we focus on the timing constraints of DAP, while assuming the semantic correctness of the aggregate functions. In DAGGTAX, 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 in parallel, after which the AVERAGE of the entire set can be computed using these two values. The computation of a holistic aggregate function cannot be decomposed into sub-aggregations. Such an example 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.

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 [27], 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 [6,39]. "**Locating**" and "**Transportation**" are optional sub-features, when these activities matter to the timing properties. The stored aggregated data may be required 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 DAP 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 [1,27]. The results of aggregation may also be accessed by a non-aggregation process, such as a control process [22].

Time-to-live. The "time-to-live" feature regulates how long the aggregated data should be preserved in the aggregator. For instance, Aurora system [1] 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, 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 2. "**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 temporally consistent in order to avoid catastrophic consequences [6]. Compared with hard real-time data, firm real-time aggregated data are useless if they are not temporally consistent [27], while soft real-time aggregated data can still be used with less value (e.g., the aggregation in the remote server [39]).

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 DAP, 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**" or "**MAXT**". 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 query [44]) or a state change in the system [6].

Real-time (P)

Real-time applications, such as automotive systems [22] and industrial monitoring systems [39], 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.

DAP 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 [16]. For a "**Firm**" real-time DAP [39], 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 [8]. "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.

5 Design Constraints and Heuristics for Data Aggregation Processes

In this section, we formulate a set of design constraints and heuristics, following the design implications imposed by the features. The design constraints 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.

5.1 Design Constraints

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 our previous work [11], we have formulated some of such dependencies as constraints in propositional formulas, which can be checked by SAT solvers. However, in the design of real-time systems, many features have numeric attributes, and are not covered by the existing constraints. For instance, deadlines and validity intervals are numeric values in time units. The execution of aggregate functions, as well as the pull and push of data, may have worst and best case response times, which can be specified in the early stage. In this section, we advance the dependency check by proposing a set of quantitative constraints for real-time properties, including timeliness of DAP, as well as absolute data validity.

Let us assume that a DAP DAP_i aggregates k raw data, denoted as RD_i^1, ..., RD_i^k, whose AVI are $RD_i^1.avi$, ..., $RD_i^k.avi$, respectively. For a raw data RD_i^m,

the worst (best) case response time for pulling the data, if applicable, is $Pull_i^m.wcrt$ ($Pull_i^m.bcrt$). The worst (best) case response time for DAP_i to execute the aggregate function is denoted as $AF_i.wcrt$ ($AF_i.bcrt$). The worst (best) case response time to push the aggregated data, if applicable, is denoted as $Push_i.wcrt$ ($Push_i.bcrt$). Further, we use $DAP_i \rightarrow DAP_j$ to denote the dependency that the aggregated data of DAP_i is used as raw data of DAP_j.

5.1.1 ***Timeliness of DAP***

For a real-time DAP, the worst case response time from the beginning of the DAP to the generation of the aggregated value, denoted as $WCRT_i$, can be calculated as:

$$WCRT_i = sum_{m=1}^{m=k}\{Pull_i^m.wcrt\} + AF_i.wcrt + Push_i.wcrt.$$

In this calculation, the worst case delay is the summation of all overheads including the pulling, aggregating and pushing data. Similarly, the best case response time $BCRT_i$ can be calculated as:

$$BCRT_i = sum_{m=1}^{m=k}\{Pull_i^m.bcrt\} + AF_i.bcrt + Push_i.bcrt.$$

The deadline of the DAP_i, denoted as $Deadline_i$, should satisfy:

$$Deadline_i \geq WCRT_i.$$

5.1.2 ***Propagation of real-time data***

Derivation of AVI. The AVI of DAP_i, denoted as $AD_i.avi$, can be derived by:

$$AD_i.avi = min_{m=1}^{m=k}\{RD_i^m.avi - Pull_i^m.wcrt - AF_i.wcrt - Push_i.wcrt\}.$$

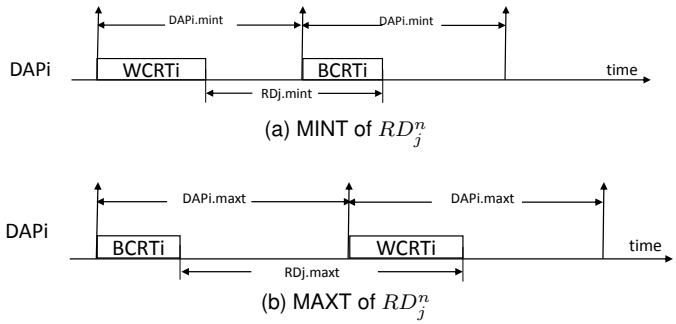
This is because the absolute validity of aggregated data is dependent on the absolute validity of its raw data. When the aggregated data is generated, the raw data has already aged for the amount of time spent on the pulling of raw data, execution of the aggregate function, and pushing of the aggregated data. Therefore, the AVI of the aggregated data should subtract these overheads from the AVI of the raw data.

Let us assume $DAP_i \rightarrow DAP_j$, that is, the aggregated data AD_i used as a real-time raw data RD_j^n. The AVI of RD_i^n is equal to the AVI of AD_i:

$$RD_j^n.avi = AD_i.avi.$$

Derivation of MINT/MAXT of raw data. Let us assume $DAP_i \rightarrow DAP_j$. Given the WCRT and BCRT of a sporadic or periodic DAP_i, we can calculate the MINT and MAXT of RD_j^n, which are actually the minimum and maximum interval of two instances of AD_i, respectively.

Fig. 5 illustrates the MINT and MAXT of RD_j^n, in case DAP_i is a sporadic process. The minimum interval occurs, when DAP_i is triggered with its minimum interval, and the first instance completes in its WCRT, while the second instance completes in its BCRT. The maximum interval occurs, on the contrary, when DAP_i is triggered with its maximum interval, while the first and second instance complete in its BCRT and WCRT, respectively. Therefore, we get the following equations:



(a) MINT of RD_j^n

(b) MAXT of RD_j^n

Fig. 5 Illustration of MINT and MAXT of RD_j^n

$$RD_j^n.mint = DAP_i.period - WCRT_i + BCRT_i.$$
$$RD_j^n.maxt = DAP_i.period - BCRT_i + WCRT_i.$$

Similarly, to derive the MINT and MAXT of RD_j^n, in case of a periodic DAP_i:

$$RD_j^n.mint = DAP_i.mint - WCDelay_i + BCDelay_i.$$
$$RD_j^n.maxt = DAP_i.maxt - BCDelay_i + WCDelay_i.$$

Worst case age of raw data. The worst case age of raw data RD_j^n when the aggregation completes, denoted as $WCAge_j^n$, can be calculated as the summation of the MAXT of RD_j^n and the worst case response time of DAP_j:

$$WCAge_j^n = RD_j^n.maxt + WCRT_j.$$

To ensure that the data are still valid, the specification must satisfy the following:

$$WCAge_j^n \leq RD_j^n.avi.$$

5.2 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 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.

Indexing for "pull" and "push" features. Appropriate indexing of data can considerably reduce the time to locate and fetch data. If data need to be pulled from or pushed to a data storage, creating an efficient indexing mechanism can reduce response times, hence improving timeliness and temporal data consistency.

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 [1], 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, with an "approximate" aggregate function. For example, Deshpande et al. introduce an approximation technique into sensor network to improve the efficiency of aggregation [18]. 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 [7]. 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 [26].

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 the 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

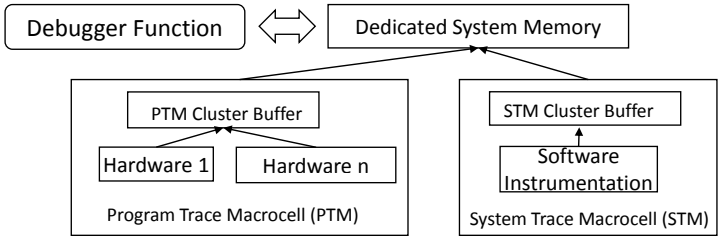


Fig. 6 General architecture of the Hardware Assisted Trace system

return the cached sub-aggregate result as an approximation [43], 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". 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 the 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.

6 Case Studies

In this section, we evaluate the usefulness of our taxonomy in aiding the design of data aggregation via two industrial projects, in Section 6.1 and Section 6.2, respectively.

6.1 Case Study I: Analyzing the Hardware Assisted Trace (HAT) Framework

In our first case study we apply DAGGTAX to analyze the design of the Hardware Assisted Trace (HAT) [61] framework, together with the engineers from Ericsson. HAT, as shown in Fig. 6, 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 DAP 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.

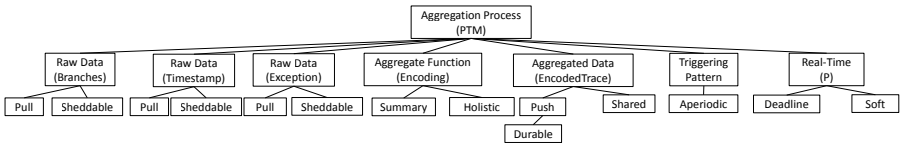


Fig. 7 The aggregation process in the PTM

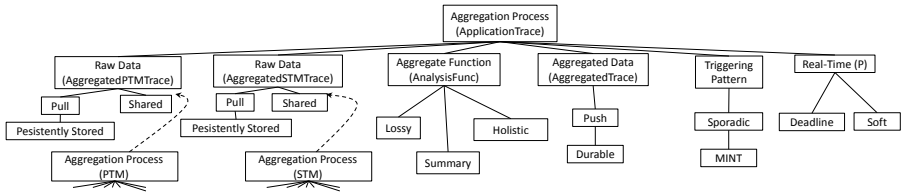
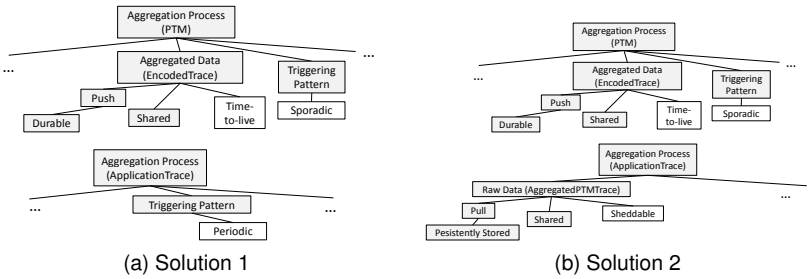


Fig. 8 The aggregation processes in the investigated HAT system

We have analyzed the features of the PTM aggregation process and the ApplicationTrace aggregation process in HAT based on our taxonomy. The diagram of the PTM aggregation process created using DAPComposer is presented in Fig. 7. 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. 8. The dashed arrows represent the data flow between DAP. 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.

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. The problem is that the data in the buffer may be overwritten before they are aggregated. It arises due to the lack of a holistic consideration on the PTM aggregation process and the ApplicationTrace aggregation process at the 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. 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.

Solutions. Considering the resource-constrained nature of the system, we and Ericsson engineers have come up with two alternative design solutions to fix this problem



(a) Solution 1 (b) Solution 2

Fig. 9 Illustration of the solutions. Unchanged features from the current design are marked in gray

based on the taxonomy. Both solutions reuse most of the features in the current design, and allow bounded buffer size.

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. 9a): (i) Instead of selecting the "aperiodic" feature, the PTM process should select "sporadic", with a defined MINT; (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; and, (iii) a "time-to-live" feature, whose value equals to the period of the ApplicationTrace process, should be added to the PTM process. 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. It also ensures that all PTM traces are aggregated by the ApplicationTrace process.

Solution 2. An alternative is to allow overwriting in a controlled manner, as illustrated in Fig. 9b. 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. 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, more efforts are required to provide real-time support, such as a real-time operating system.

6.2 Case Study II: Timing Analysis for a Brake-By-Wire System

We illustrate the analysis of high-level timing specifications during DAP design, via a Brake-By-Wire (BBW) system [32]. A BBW system is part of an automotive system that controls the torques for the wheels in case the brake pedal is pressed by the driver. Both the wheel speed and the press of pedal are monitored by respective sensors, and transmitted to the computer in the vehicle via network. The braking torque

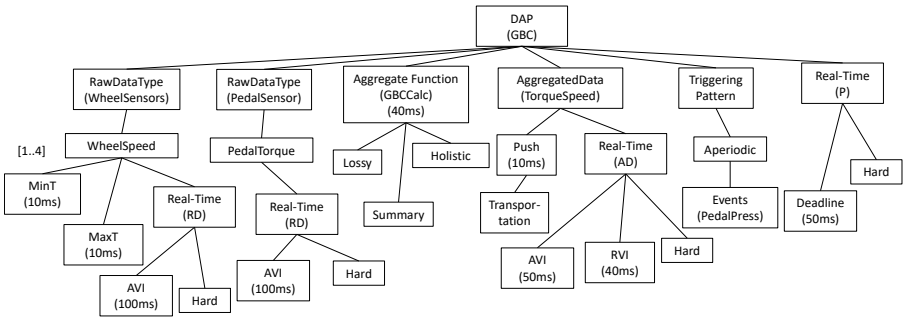


Fig. 10 DAP in the Global Brake Controller (GBC) of the BBW system

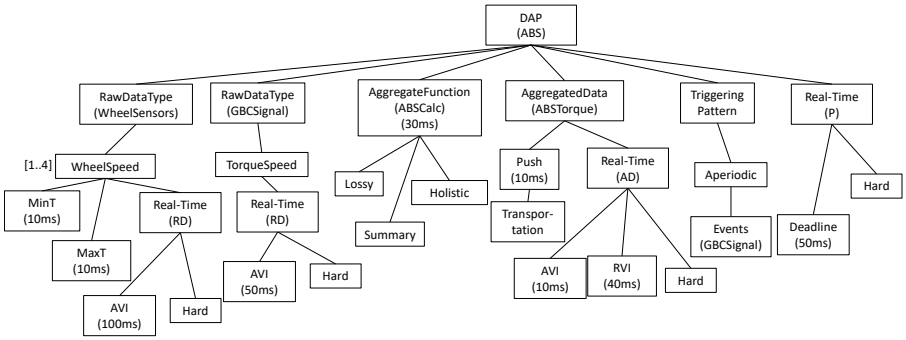


Fig. 11 DAP in the Anti-lock Braking System (ABS) of the BBW system

is then calculated using these data, and sent to the actuator for braking. In order for the braking to take place in the right moment and guarantee safety, stringent timing requirements must be enforced on the DAP in the system. The processes must meet their deadlines, while the data used by the calculations have to be fresh. Otherwise, traffic accidents could occur. Therefore, validating the correctness of the DAP specifications with respect to the timing properties is crucial.

We identify two essential DAP in this BBW system. The first DAP, presented in Fig. 10, is performed in the Global Brake Controller (GBC). Triggered by the press on the brake pedal, the GBC calculates the estimated vehicle speed and torque, by aggregating the wheel speed from four wheel speed sensors, together with the requested torque from the pedal sensor. The results of the GBC DAP are sent to the Anti-lock Braking System (ABS) controller, which performs another DAP (Fig. 11) that calculates the actual braking torque by aggregating the estimated vehicle speed and torque, with the current wheel speed. DAGGTAX provides a structured way to specify these DAP, especially their timing constraints, shown in Fig. 10 and Fig. 11.

In addition, we apply the formulas in Section 5.1 to check whether the specification is consistent. By applying the calculation in Section 5.1.1, we obtain that the worst-case response times of GBC-DAP and ABS-DAP are 50ms and 40ms, respectively. Since their deadlines are both 50ms, this specification satisfies the timeliness requirements. We also apply the formulas in Section 5.1.2 to calculate the validity intervals and ages of the propagated data. One important piece of data is the wheel

speed. We calculate that the worst case age of the wheel speed data is 90ms during the entire GBC and ABS DAP. Since the AVI of wheel speed is 100ms, we can conclude that this specification satisfies temporal data validity for the wheel speed data.

6.3 Summary

We have implemented a tool called SAFARE (SAt-based Feature-oriented dAta aggREgation design) [11] for the DAGGTAX-based specification of data aggregation processes. SAFARE provides a graphical interface to selecting the features, and integrates a SAT solver to check whether any design constraints are violated. In addition to the aforementioned case studies, we have also applied DAGGTAX and SAFARE to the design of a cloud-monitoring system [11].

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, which provides help in both identifying reusable DAP and constructing new DAP. 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 reduces costs. Design solutions can be constructed by composing reusable features, and reasoned about based on the taxonomy, which contributes to a reduced design space.

7 Related Work

Many researchers have promoted the understanding of data aggregation on various aspects. Among them, considerable effort has been dedicated to the study of aggregate functions. Mesiar et al. [46], Marichal [30], and Rudas et al. [52] 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. [52]. In order to design a software system that computes aggregation efficiently, Gray et al. [25] 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. [43] have extended Gray's taxonomy, and classified aggregate functions according to their state requirements, tolerance of loss, duplicate sensitivity, and monotonicity. Fasolo et al. [20] classify aggregate functions with respect to four dimensions, which are lossy aggregation, duplicate sensitivity, resilience to losses/failures and correlation awareness. Jesus et al. [31] formally define and characterize different types of aggregate functions, and also organize the relevant techniques for aggregation as a taxonomy. Our taxonomy builds on such work that focuses on the aggregate functions mainly, and provide a comprehensive view of the entire aggregate processes instead.

Summarizability is important to the correctness of data aggregation, that is, the selected aggregate function can yield the correct aggregated results from the target raw data [50]. To analyze summarizability, Lenz et al. [41] have classified raw data into "stock", "flow" and "value-per-unit", based on the semantics of the raw

data. They have also classified aggregate functions into "summarizable" and "non-summarizable", which is based on the same criteria as the "progressive" and "holistic" categorization in DAGGTAX. Based on their classifications, conditions are proposed to check the summarizability of the aggregation, that is, whether the aggregate function is suitable for the raw data. For instance, an aggregation that generates the total population of a state by adding up the populations of a list of areas is not summarizable, if the list of areas is not complete. Niemi et al. [47] have further proposed more aggregation types, as well as rules to detect their summarizability. These classifications emphasize the semantics of data and the aggregate functions, and hence are important for deciding the correct aggregate functions to achieve the desired goals of data analysis. Our taxonomy, on the contrary, assumes that summarizability has already been analyzed and achieved. Our work focuses on the next step, that is, to design the process that performs aggregation using the decided aggregate function, such that the steps of the aggregation process are well understood, and the desired timing constraints can be satisfied.

A large proportion of existing work has its 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 [20]. Besides a classification of aggregate functions that we have discussed in the previous paragraph, Fasolo et al. [20] 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. [56], Makhloufi et al. [45], and Rajagopalan [51], with respect to different classification criteria. A more recent work is proposed by Pourghebleh and Navimipour [49], which classifies the data aggregation architectures and mechanisms in the Internet-of-Things. In contrast to the aforementioned work that focuses mainly on aggregation protocols, Alzaid et al. [2] have proposed a taxonomy of secure aggregation schemes that classifies them into different models. Sirsikar and Anavatti [55] have investigated and classified the common issues in in-network aggregation. All the existing related work 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.

8 Conclusions and Future Work

In this paper, we have investigated the characteristics of data aggregation processes in a variety of applications, with a particular focus on the real-time properties. The survey has inspired a taxonomy of DAP called DAGGTAX, which presents the common and variable characteristics 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. In addition, a set of design constraints and heuristics have been proposed, which can reduce the design space and guide the realization of the selected features, so that the timing constraints can be satisfied. The usefulness of the taxonomy has

been demonstrated on two industrial case studies. Flaws can be identified at design time, and solutions can be proposed at design level, by applying the taxonomy to the analysis.

DAGGTAX covers all the constituents of a DAP (raw data, aggregate function, and aggregated data), the activities in the general steps of a DAP, as well as timing properties that are widely considered crucial for real-time data management. The specific activities that are involved in the DAP and have impact on the timing properties, such as transportation and transformation of data, are also represented as features in DAGGTAX. However, we cannot claim that DAGGTAX covers all possible features of DAP in real-time applications, as some applications may include unique activities within the DAP that greatly affect timing, or particular properties such as distribution of triggering time are considered of special interest for some systems. In this case, DAGGTAX can serve as a basis, as it includes the basic constituents and steps of DAP, to develop an evolved feature model incorporating the particular features of the special case.

Our taxonomy can be viewed as a framework for analyzing the dependencies between features and between DAP. Our future work aims to derive more constraints that can guide the design of DAP. In addition, a tool that automates the reasoning will be useful for designers, which we have started in [11]. In the future, more advanced analysis techniques, such as model checking and schedulability analysis, could be integrated to detect conflicting features and provide guidance for trade-offs.

References

1. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: A new model and architecture for data stream management. The VLDB Journal 12(2), 120–139 (2003)
2. Alzaid, H., Foo, E., Nieto, J.M.G., Park, D.: A taxonomy of secure data aggregation in wireless sensor networks. International Journal of Communication Networks and Distributed Systems 8(1-2), 101–148 (2012)
3. Arai, B., Das, G., Gunopulos, D., Kalogeraki, V.: Approximating aggregation queries in peer-to-peer networks. In: Proceedings of the 22nd International Conference on Data Engineering. pp. 42–42 (2006)
4. Babcock, B., Datar, M., Motwani, R.: Load shedding for aggregation queries over data streams. In: Proceedings of the 20th International Conference on Data Engineering. pp. 350–361 (2004)
5. Bar, A., Casas, P., Golab, L., Finamore, A.: Dbstream: An online aggregation, filtering and processing system for network traffic monitoring. In: Proceedings of the 2014 International Wireless Communications and Mobile Computing Conference. pp. 611–616 (2014)
6. Baulier, J., Blott, S., Korth, H.F., Silberschatz, A.: A database system for real-time event aggregation in telecommunication. In: Proceedings of the 24rd International Conference on Very Large Data Bases. pp. 680–684 (1998)
7. Botan, I., Fischer, P.M., Kossmann, D., Tatbul, N.: Transactional stream processing. In: Proceedings of the 15th International Conference on Extending Database Technology. pp. 204–215 (2012)
8. Bür, K., Omiyi, P., Yang, Y.: Wireless sensor and actuator networks: Enabling the nervous system of the active aircraft. IEEE Communications Magazine 48(7), 118–125 (2010)
9. Buttazzo, G.C.: Hard real-time computing systems: predictable scheduling algorithms and applications, vol. 24. Springer Science & Business Media (2011)
10. Cai, S., Gallina, B., Nyström, D., Seceleanu, C.: Daggtax: a taxonomy of data aggregation processes. In: International Conference on Model and Data Engineering. pp. 324–339. Springer (2017)
11. Cai, S., Gallina, B., Nyström, D., Seceleanu, C., Larsson, A.: Tool-supported design of data aggregation processes in cloud monitoring systems. Journal of Ambient Intelligence and Humanized Computing (Feb 2018)

12. Chaudhuri, S., Dayal, U.: An overview of data warehousing and olap technology. SIGMOD Record 26(1), 65–74 (1997)
13. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software process: Improvement and practice 10(1), 7–29 (2005)
14. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice 10(2), 143–169 (2005)
15. Czarnecki, K., Ulrich, E.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
16. Defude, B., Delot, T., Ilarri, S., Zechinelli, J.L., Cenerario, N.: Data aggregation in vanets: The vespa approach. In: Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services. pp. 13:1–13:6 (2008)
17. Demiris, G., Hensel, B.K.: Technologies for an aging society: a systematic review of âĂIJsmart homeâĂİ applications. Yearbook of medical informatics 17(01), 33–40 (2008)
18. Deshpande, A., Guestrin, C., Madden, S.R., Hellerstein, J.M., Hong, W.: Model-driven data acquisition in sensor networks. In: Proceedings of the 13th International Conference on Very Large Data Bases. pp. 588–599 (2004)
19. Eichler, S., Merkle, C., Strassberger, M.: Data aggregation system for distributing inter-vehicle warning messages. In: Proceedings of the 39th Annual IEEE Conference on Local Computer Networks. pp. 543–544 (2006)
20. Fasolo, E., Rossi, M., Widmer, J., Zorzi, M.: In-network aggregation techniques for wireless sensor networks: a survey. IEEE Wireless Communications 14(2), 70–87 (2007)
21. Golab, L., Johnson, T., Seidel, J.S., Shkapenyuk, V.: Stream warehousing with datadepot. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. pp. 847–854 (2009)
22. Goud, G., Sharma, N., Ramamritham, K., Malewar, S.: 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. pp. 335–341 (2006)
23. Graefe, G.: Query evaluation techniques for large databases. ACM Computing Surveys (CSUR) 25(2), 73–169 (1993)
24. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers Inc., 1st edn. (1992)
25. Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Mining and Knowledge Discovery 1(1), 29–53 (1997)
26. Gürgen, L., Roncancio, C., Labbé, C., Olive, V.: Transactional issues in sensor data management. In: Proceedings of the 3rd Workshop on Data Management for Sensor Networks. pp. 27–32 (2006)
27. He, T., Gu, L., Luo, L., Yan, T., Stankovic, J., Son, S.: An overview of data aggregation architecture for real-time tracking with sensor networks. In: Proceedings of the 20th International Parallel and Distributed Processing Symposium. pp. 8 pp.– (2006)
28. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. SIGMOD Record 26(2), 171–182 (1997)
29. Iftikhar, N.: 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. pp. 14–19 (2009)
30. Jean-Luc, M.: Aggregation functions for decision making. In: Decision Making Process: Concepts and Methods, chap. 17, pp. 673–721. Wiley-Blackwell
31. Jesus, P., Baquero, C., Almeida, P.S.: A survey of distributed data aggregation algorithms. IEEE Communications Surveys & Tutorials 17(1), 381–404 (2015)
32. Kang, E.Y., Enoiu, E.P., Marinescu, R., Seceleanu, C., Schobbens, P.Y., Pettersson, P.: A methodology for formal analysis and verification of east-adl models. Reliability Engineering & System Safety 120, 127–138 (2013)
33. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (foda) feasibility study. Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (1990), http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231
34. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: Form: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering 5(1), 143–168 (1998)
35. Karkouch, A., Mousannif, H., Al Moatassime, H., Noel, T.: Data quality in internet of things: A state-of-the-art survey. Journal of Network and Computer Applications 73, 57–81 (2016)
36. Kitchin, R.: The real-time city? big data and smart urbanism. GeoJournal 79(1), 1–14 (2014)

37. Krishnamurthy, S., Wu, C., Franklin, M.: On-the-fly sharing for streamed aggregation. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. pp. 623–634 (2006)
38. Kulik, J., Heinzelman, W., Balakrishnan, H.: Negotiation-based protocols for disseminating information in wireless sensor networks. Wireless networks 8(2/3), 169–185 (2002)
39. Lee, A.N., Lastra, J.L.M.: Data aggregation at field device level for industrial ambient monitoring using web services. In: Proceedings of the 9th IEEE International Conference on Industrial Informatics. pp. 491–496. IEEE (2011)
40. Lee, J.: Smart factory systems. Informatik-Spektrum 38(3), 230–235 (2015)
41. Lenz, H.J., Shoshani, A.: Summarizability in olap and statistical data bases. In: Proceedings of the 9th Scientific and Statistical Database Management. pp. 132–143 (1997)
42. Lopez, I.F.V., Snodgrass, R.T., Moon, B.: Spatiotemporal aggregate computation: A survey. IEEE Transactions on Knowledge and Data Engineering 17(2), 271–286 (2005)
43. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: A tiny aggregation service for ad-hoc sensor networks. ACM SIGOPS Operating Systems Review 36(SI), 131–146 (2002)
44. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: An acquisitional query processing system for sensor networks. ACM Transactions on Database Systems 30(1), 122–173 (2005)
45. Makhloufi, R., Doyen, G., Bonnet, G., Gaïti, D.: A survey and performance evaluation of decentralized aggregation schemes for autonomic management. International Journal of Network Management 24(6), 469–498 (2014)
46. Mesiar, R., Kolesárová, A., Calvo, T., Komorníková, M.: A review of aggregation functions. In: Fuzzy Sets and Their Extensions: Representation, Aggregation and Models, vol. 220, pp. 121–144. Springer Berlin Heidelberg (2008)
47. Niemi, T., Niinimäki, M., Thanisch, P., Nummenmaa, J.: Detecting summarizability in olap. Data & Knowledge Engineering 89, 1–20 (2014)
48. Oyamada, M., Kawashima, H., Kitagawa, H.: Data stream processing with concurrency control. SIGAPP Applied Computing Review 13(2), 54–65 (2013)
49. Pourghebleh, B., Navimipour, N.J.: Data aggregation mechanisms in the internet of things: A systematic review of the literature and recommendations for future research. Journal of Network and Computer Applications 97, 23–34 (2017)
50. Rafanelli, M., Shoshani, A.: Storm: A statistical object representation model. In: International Conference on the 5th Scientific and Statistical Database Management. pp. 14–29. Springer (1990)
51. Rajagopalan, R., Varshney, P.: Data-aggregation techniques in sensor networks: A survey. IEEE Communications Surveys Tutorials 8(4), 48–63 (2006)
52. Rudas, I.J., Pap, E., Fodor, J.: Information aggregation in intelligent systems: An application oriented approach. Knowledge-Based Systems 38, 3–13 (2013)
53. Santana, E.F.Z., Chaves, A.P., Gerosa, M.A., Kon, F., Milojicic, D.S.: Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. ACM Computing Surveys (CSUR) 50(6), 78 (2017)
54. Schweppe, H., Zimmermann, A., Grill, D.: Flexible on-board stream processing for automotive sensor data. IEEE Transactions on Industrial Informatics 6(1), 81–92 (2010)
55. Sirsikar, S., Anavatti, S.: Issues of data aggregation methods in wireless sensor network: a survey. Procedia Computer Science 49, 194–201 (2015)
56. Solis, I., Obraczka, K.: In-network aggregation trade-offs for data collection in wireless sensor networks. International Journal of Sensor Networks 1(3-4), 200–212 (2006)
57. Song, X., Liu, J.: How well can data temporal consistency be maintained? In: Proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design (CACSD). pp. 275–284 (1992)
58. Srivastava, D., Dar, S., Jagadish, H.V., Levy, A.Y.: Answering queries with aggregation using views. In: Proceedings of the 22th International Conference on Very Large Data Bases. pp. 318–329 (1996)
59. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming 79, 70–85 (2014)
60. Vaisman, A., Zimányi, E.: Data Warehouse Systems: design and implementation. Springer, 1st edn. (2014)
61. Vitucci, C., Larsson, A.: Hat, hardware assisted trace: Performance oriented trace & debug system. In: Proceedings of 26th International Conference on Software & Systems Engineering and their Applications (2015)
62. Yun, X., Wu, G., Zhang, G., Li, K., Wang, S.: Fastraq: A fast approach to range-aggregate queries in big data environments. IEEE Transactions on Cloud Computing 3(2), 206–218 (2015)