DAGGTAX: A Taxonomy of Data Aggregation Processes

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

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

Abstract. Data aggregation processes are essential constituents for data management in modern computer systems, such as decision support systems and Internet of Things (IoT) systems. Due to the heterogeneity and real-time constraints in such systems, designing appropriate data aggregation processes often demands considerable effort. A study on the characteristics of data aggregation processes is then desirable, as it provides a comprehensive view of such processes, potentially facilitating their design, as well as the development of tool support to aid designers. In this paper, we propose a taxonomy called DAGGTAX, which is a feature diagram that models the common and variable characteristics of data aggregation processes, with a special focus on the real-time aspect. The taxonomy can serve as the foundation of a design tool, which we also introduce, enabling designers to build an aggregation process by selecting and composing desired features, and to reason about the feasibility of the design. We apply DAGGTAX on industrial case studies, showing that DAGGTAX not only strengthens the understanding, but also facilitates the model-driven design of data aggregation processes.

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 [21], to save storage space [25], or to reduce bandwidth and energy costs [17]. 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 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 [19], avionic systems [8] and industrial automation [29] 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 [36]. We consider a DAP as a sequence of three ordered activities that allow raw data to be transformed into aggregated data via an aggregate function. First, a DAP starts with preparing the raw data needed for the aggregation from the data source into the aggregation unit called the aggregator. Next, an aggregate function is applied by the aggregator on the raw data, and produces the aggregated data. Finally, the aggregated data may be further handled by the aggregator, for example, to be saved into storage or provided to other processes. The main constituents of these activities are the raw data, the aggregate function and the aggregated data.

The main contribution of this paper is a high-level taxonomy of data aggregation processes, called DAGGTAX, presented as a feature diagram [26]. The aim of our taxonomy is to ease the design of aggregation processes, by providing a comprehensive view on the features and cross-cutting constraints, by a systematic representation. The intuition is that, to design a DAP, we must understand the desired features of its main constituents, as well as those of the DAP itself. Such features, ranging from functional features (such as data sharing) to extra-functional features (such as timeliness), are varying depending on different applications. One aspect of the understanding is to distinguish the mandatory features from the optional ones, so that the application designer is able to sort out the design choices. Another aspect is to comprehend the implications of the features, and to reason about the (possible) impact on one another. Conflicts may arise among features, in that the existence of one feature may prohibit another one. 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. The proposed taxonomy can serve as a basis for such automated reasoning. To evaluate the usefulness of DAGGTAX, we have developed a DAGGTAX-based tool called DAPComposer, which enables constructing DAP by selecting the desired features, and have applied it on two industrial case studies. The evaluations demonstrate that DAGGTAX raises the awareness of design issues in DAP, and helps to reason about possible trade-offs between different design solutions.

The remainder of the paper is organized as follows. In Section 2 we discuss the existing taxonomies of data aggregation. In Section 3 we provide background information. Section 4 presents our proposed taxonomy, followed by the tool and the case studies in Section 5. In Section 6 we conclude the paper and outline possible future works.

2 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. [34], Marichal [33], and Rudas et al. [36] 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. [36]. In order to design a software system that computes aggregation efficiently, Gray et al. [21] 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. [30] have extended Gray's taxonomy, and classified aggregate functions according to their state requirements, tolerance of loss, duplicate sensitivity, and monotonicity. Fasolo et al. [17] classify aggregate functions with respect to four dimensions, which are lossy aggregation, duplicate sensitivity, resilience to losses/failures and correlation awareness. Our taxonomy builds on such work that focuses on the aggregate functions mainly, and provide a comprehensive view of the entire aggregate processes instead.

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 [17]. Besides a classification of aggregate functions that we have discussed in the previous paragraph, Fasolo et al. [17] 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. [38], Makhloufi et al. [32], and Rajagopalan [35], with respect to different classification criteria. 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. 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.

3 Background

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

3.1 Timeliness and Temporal Data Consistency

In a real-time system, the correctness of a computation depends on both the logical correctness of the results, and the time at which the computation completes [9]. The property of completing the computation by a given deadline is referred to as *timeliness*. A real-time task can be classified as *hard*, *firm* or *soft* real-time, depending on the consequence of a deadline miss [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 can be seen as aperiodic.

Real-time applications often monitor the state of the environment, and react to changes accordingly and timely. The environment state is represented as data in the system, which must be updated according to the actual environment state. The coherency between the value of the data in the system and its corresponding environment state is referred to as *temporal data consistency*, which includes two aspects, the *absolute temporal validity* and *relative temporal validity* [39]. A data instance is absolute valid, if the timespan between the time of sampling its corresponding real-world value, and the current time, is less than a specified *absolute validity interval*. A data instance derived from a set of data instances (base data) is absolute valid if all participating base data are absolute valid. A derived data instance is relative valid, if the base data are sampled within a specified interval, called *relative validity interval*.

Data instances that are not temporally consistent may lead to different consequences. Different levels of strictness with respect to temporal consistency thus exist, which are *hard*, *firm* and *soft* real-time, in a decreasing order of strictness. Using outdated hard real-time data could cause disastrous consequences, and therefore this should not appear. Firm real-time data are useless if they are outdated, whereas outdated soft real-time data can still be used, but will yield degraded usefulness.

3.2 Feature Model and Feature Diagram

The notion of *feature* was first introduced by Kang et al. in the Feature-Oriented Domain Analysis (FODA) method [26], in order to capture both the common characteristics of a family of systems as well as the differences between individual systems. Kang et al. define a feature as a prominent or distinctive system characteristic visible to end-users. Czarnecki and Eisenecker extend the definition of a feature to be any functional or extra-functional characteristic at the requirement, architecture, component, or any other level [13]. 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* [26], 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≥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.

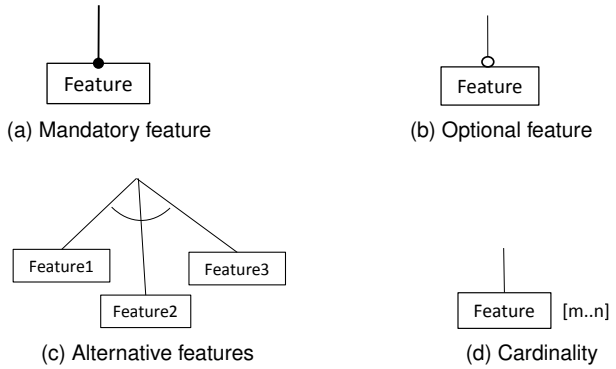

Fig. 1. Notations of a feature diagram

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 [40], software designers can create feature diagrams using a rich graphic interface. Designers can specify constraints across features as well as models, to ensure that only valid configurations are generated from the feature diagram.

4 Our Proposed Taxonomy

In this section we propose a taxonomy of data aggregation processes, called DAGGTAX, as an ordered arrangement of common and variable features revealed by the survey in our referred report [10]. The taxonomy for these common and variable characteristics not only leads to a clear understanding of the aggregation process, but also lays a solid foundation for an eventual tool support for analyzing the impact of different features on the design. In this paper, we also present an initial version of such tool.

We choose feature diagrams as the presentation of our taxonomy, mainly due to two reasons. First, features may be used to model both functional and extra-functional characteristics of systems. This allows us to capture cross-cutting aspects that have impact on multiple software modules related to different concerns. Second, the notation of feature diagrams is simple to construct, powerful to capture the common and variable characteristics of different data aggregation processes, and intuitive to provide an organizational view of the processes. The taxonomy is shown in Fig. 2.

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. 2 from the top-level features under "Aggregation

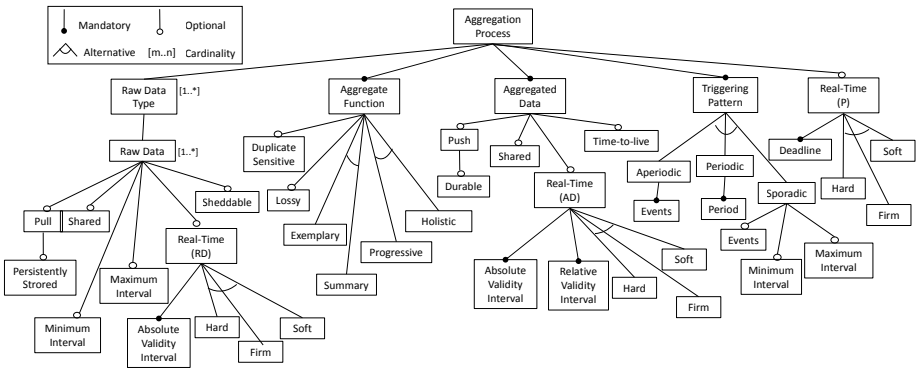

Fig. 2. The taxonomy of data aggregation processes

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.

4.1 Raw Data

One of the mandatory features of real-time data aggregation is the raw data involved in the process. Raw data are the data provided by the DAP data sources. One DAP may involve one or more types of raw data. The multiplicity is reflected by the cardinality [1..*] next to the feature "**Raw Data Type**" in Fig. 2. 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. 3a. For instance, a traditional DBMS adopts the pull scheme, in which raw data are acquired from disks using SQL queries and aggregated in the main memory. "Pull" is considered to be an optional feature of raw data, since not every DAP pulls data actively from data source. If raw data have the "pull" feature, pulling raw data actively from the data source is a necessary part of the aggregation process, including the selection of data as well as the shipment of data from the data source. If the raw data do not have the "pull" feature, they are pushed into the aggregator (Fig. 3b). 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.

An optional sub-feature of "Pull" 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.

Fig. 3. Raw data acquisition schemes

Shared Raw data of a DAP may be read or updated by other processes at the same time when they are read for aggregation [25]. 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.

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 [30], 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)" in Section 4.3) and the process ("Real-Time (P)" in Section 4.5).

Raw data with real-time constraints are classified as "**hard**", "**firm**" or "**soft**" real-time, depending on the strictness with respect to temporal consistency. They are represented as alternative sub-features of the real-time feature. As we have explained in Section 3, hard real-time data (such as sensor data from a field device [29]) and firm real-time data (such as surveillance data [23]) 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 [23]).

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 the surveillance system VigilNet [23], 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.

4.2 Aggregate Function

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

Duplicate Sensitive "Duplicate sensitivity" has been introduced as a dimension by Madden et al. [30] and Fasolo et al. [17]. An aggregate function is duplicate sensitive, if an incorrect aggregated result is produced due to a duplicated raw data. For example, COUNT, which counts the number of raw data instances, is duplicate sensitive, since a duplicated instance will lead to a result one bigger than it should be. MIN, which returns the minimum value of a set of instances, is not duplicate sensitive because its result is not affected by a duplicated instance. "Duplicate sensitive" is considered as an optional feature of the aggregate function.

Exemplary or Summary According to Madden et.al [30], 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 of values .

Lossy An aggregate function is "lossy", if the raw data cannot be reconstructed from the aggregated data alone [17]. For example, SUM, which computes the summation of a set of raw data instances, is a lossy function, as one cannot reproduce the raw data instances from the aggregated summation value without any additional information. On the contrary, a function that concatenates raw data instances with a known delimiter is not lossy, since the raw data can be reconstructed by splitting the concatenation. Therefore, we introduce "lossy" as an optional feature of aggregate functions.

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

4.3 Aggregated Data

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

Push In some survey DAP examples, sending aggregated data to another unit of the system is an activity of the aggregator immediately after the computation of aggregation. This is considered as an active step of the aggregation process, and is represented by the feature "push". For example, in the group layer aggregation of VigilNet [23], 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, 29]. The stored aggregated data may be required to be durable, which means that the aggregated data must survive potential system failures. Therefore, "**durable**" is considered as an optional sub-feature of the "push" feature.

Shared Similar to raw data, the aggregated data has an optional "shared" feature too, to represent the characteristic of some of the surveyed 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, 23]. The results of aggregation may also be accessed by a non-aggregation process, such as a control process [19].

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 3. "**Absolute validity interval**" and "**relative validity interval**" are two mandatory sub-features of the "Real-Time (AD)" feature.

Similar to raw data, the real-time feature of aggregated data has "**hard**", "**firm**" and "**soft**" as alternative sub-features. If the aggregated data are required to be hard real-time, they have to be ensured 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 [23], while soft real-time aggregated data can still be used with less value (e.g., the aggregation in the remote server [29]).

4.4 Triggering Pattern

"Triggering pattern" refers to how the DAP is activated, which is a mandatory feature. We consider three types of triggering patterns for the activation of 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 [31]) or a state change in the system [6].

4.5 Real-time (P)

Real-time applications, such as automotive systems [19] and industrial monitoring systems [29], 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.

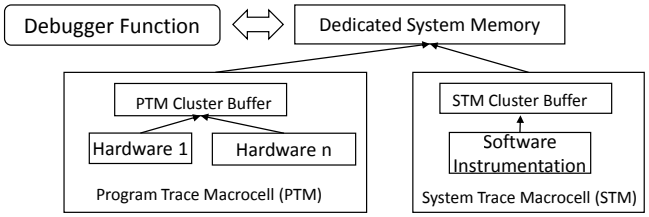

Fig. 4. General architecture of the Hardware Assisted Trace system

Aggregation processes may have different types of timeliness constraints, depending on the consequences of missing their deadlines. For a **soft** real-time DAP, a deadline miss will lead to a less valuable aggregated result [14]. For a **firm** real-time DAP [29], 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 Case Studies

In this section, we evaluate the usefulness of our taxonomy in aiding the design of data aggregation via two industrial projects together with the engineers from Ericsson, in Section 5.1 and Section 5.2, respectively. Prior to the case studies we have implemented a tool called DAPComposer (Data Aggregation Process Composer) in Javascript. The tool provides a graphical user interface for designers to create DAP, by simply enabling/disabling features from DAGGTAX. More details of the tool are included in the technical report [10]. Although to date only primitive constraints intrinsic to the feature model are checked by DAPComposer, we plan to mature the tool with more sophisticated analysis capabilities, such as timing analysis, in the next version.

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

In our first case study we apply DAGGTAX to analyze the design of the Hardware Assisted Trace (HAT) [41] framework. HAT, as shown in Fig. 4, is a framework for debugging functional errors in an embedded system. In this framework, a debugger function runs in the same system as the debugged program, and collects both hardware and software run-time traces continuously. Together with the engineers we have analyzed the aggregation processes in their current design. At a lower level, a Program Trace Macrocell (PTM) aggregation process aggregates traces from hardware. These aggregated PTM traces, together with software instrumentation traces from the System Trace

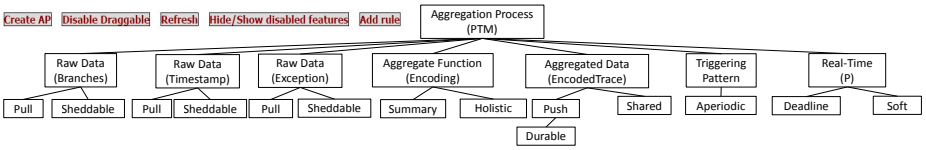

Fig. 5. The aggregation process in the PTM

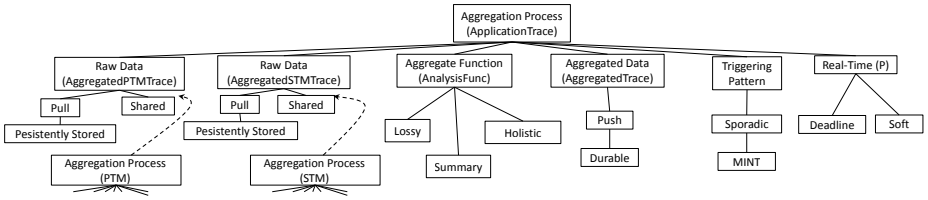

Fig. 6. The aggregation processes in the investigated HAT system

Macrocell (STM), are then aggregated by a higher level ApplicationTrace aggregation process, to create an informative trace for the debugged application.

We have analyzed the features of the PTM aggregation process and the ApplicationTrace aggregation process in HAT based on our taxonomy. The diagram of the PTM aggregation process created using DAPComposer is presented in Fig. 5. 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. 6. 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 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.

We have provided two solutions to solve the identified problem. Due to lack of space, the details of this case study is included in the technical report [10].

5.2 Case Study II: A Cloud Monitoring System for Enhanced Auto-scaling

In our second case study we apply DAGGTAX to design a cloud-monitoring system that enables auto scaling based on both virtual-machine-level and application-level performance measurements, by extending the open-source OpenStack framework, which collects measurements only from the virtual-machine level. DAGGTAX is applied to both the existing framework, as well as the new design. Based on the feature diagrams, we analyze the pros and cons of different feature combinations, and decide the design solution. Some features and DAP of the existing framework are identified as reusable, and reused in the solution. We refer the details of this case study to the paper [11].

5.3 Summary

The engineers in the evaluation acknowledge that our taxonomy bridges the gap between the properties of data and the properties of the process, which has not been elaborated by other taxonomies. Our taxonomy enhances the understanding of the system by structuring the common and variable features of data aggregation processes, 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. Due to these benefits, the engineers see great value in a more mature version of DAPComposer for data aggregation applications, based on our taxonomy.

6 Conclusions and Future Work

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

Our taxonomy can be viewed as a framework for analyzing the dependencies between features and between DAP. Our future work aims to integrate more advanced analysis techniques, such as model checking and schedulability analysis, to detect conflicting features and provide guidance for trade-offs. These techniques can be integrated into DAPComposer. We also plan to extend the DAPComposer so that users can specify their constraints and validate the design through the tool automatically.

Acknowledgment This work is funded by the Knowledge Foundation of Sweden (KK-stiftelsen) within the DAGGERS project. We acknowledge Alf Larsson, Andreas Ermedahl and Carlo Vitucci from Ericsson Research for their help in the case studies.

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. Communications Magazine, IEEE 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. Tech. rep. (2016), `http://www.es.mdh.se/publications/4628-`
11. Cai, S., Gallina, B., Nyström, D., Seceleanu, C.: Design of cloud monitoring systems via daggtax: a case study. In: The 8th International Conference on Ambient Systems, Networks and Technologies (May 2017)
12. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software process: Improvement and practice 10(1), 7–29 (2005)
13. Czarnecki, K., Ulrich, E.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
14. 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)
15. 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)
16. 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)
17. Fasolo, E., Rossi, M., Widmer, J., Zorzi, M.: In-network aggregation techniques for wireless sensor networks: a survey. Wireless Communications, IEEE 14(2), 70–87 (2007)
18. 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)

19. 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)
20. Graefe, G.: Query evaluation techniques for large databases. ACM Comput. Surv. 25(2), 73–169 (1993)
21. 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)
22. 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
23. 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)
24. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. SIGMOD Rec. 26(2), 171–182 (1997)
25. 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)
26. 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
27. 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)
28. Kulik, J., Heinzelman, W., Balakrishnan, H.: Negotiation-based protocols for disseminating information in wireless sensor networks. Wirel. Netw. 8(2/3), 169–185 (2002)
29. 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)
30. 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)
31. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: An acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30(1), 122–173 (2005)
32. 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)
33. Marichal, J.L.: Aggregation Functions for Decision Making, pp. 673–721. ISTE (2010)
34. 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)
35. Rajagopalan, R., Varshney, P.: Data-aggregation techniques in sensor networks: A survey. Communications Surveys Tutorials, IEEE 8(4), 48–63 (2006)
36. Rudas, I.J., Pap, E., Fodor, J.: Information aggregation in intelligent systems: An application oriented approach. Knowledge-Based Systems 38, 3–13 (2013)
37. Schweppe, H., Zimmermann, A., Grill, D.: Flexible on-board stream processing for automotive sensor data. Industrial Informatics, IEEE Transactions on 6(1), 81–92 (2010)
38. 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)
39. 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. pp. 275–284 (1992)
40. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Sci. Comput. Program. 79, 70–85 (2014)
41. 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)